6502 USER NOTES

NO.17 $2.50

6502 FORTH is here !! (SEE INSIDE BACK COVER)

EDITORIAL

This will be the last regular issue of 6502 User Notes as we all know it.

We won't be disappearing altogether, however, just merging with a new magazine called COMPUTE. In fact, you'll be receiving the January issue of COMPUTE instead of #18 of the User Notes as the last issue of this subscription.

The decision to merge with COMPUTE was arrived at only after long deliberation on the future of the 'Notes' and its purpose.

Obviously, I feel that this is the best way to keep some continuity in the support of KIM and KIM-derived products.

I'll be writing a column for COMPUTE magazine so don't look for me to leave the 6502 arena all that quickly.

Actually, I just need some free time for all those personal projects which have been stacking up for a time (noise generation (music?), computerized bio-feedback, several hardware designs, a box for my system, etc.)

You should pat yourself on the back for being a big part of what this publication has become. I thank you.

I haven't mentioned that I am now living in California. Yep, moved again! Am working for Rockwell as their-you guess it!-newsletter editor. Thought that would grab you!!!

WHAT ELSE IS NEW?

HDE BASIC. As I've reported previously, HDE now has the source code rights to Microsoft BASIC. Well, about a week ago, I received an interim version of HDE BASIC for my comments. Several significant additions have been made to BASIC which really improves its operation. I guess the addition of a line editor really impressed me the most. As you may recall, the lack of a line editor was one of my biggest gripes. The line editor in HDE BASIC operates in the same manner as the HDE Text Editor (TED). This means that only one method of line editing need be learned. In HDE BASIC lines may be moved, appended to and copied. Binary files can be loaded from disk under program control which makes linking to machine language a snap. A command is included that not only appends a program from disk to a memory resident program, but also resequences the line numbers in the appended program to avoid duplicate line numbers. (A very neat trick that I haven't seen in any other version of BASIC). HDE BASIC also supports data files of the same type as MICRO-Z BASIC. There is no facility in the interim version of HDE BASIC for setting up input and output files, but that will be added before the program is released. HDE expects to be offering a full disk BASIC early in 1980.

6502 USER NOTES is published bimonthly by Eric C. Rehnke (540 S Ranch View Cir #61, Anaheim Hills, Ca 92807 (714-637-4686)). Subscription rates for Volume 3 are $13.00 (US and Canada) and $19.00 elsewhere. One subscription includes all 6 issues of a volume and cannot carry over part way into the next volume. If less than a full volume is desired, remit $2.50 for each issue wanted.

BASIC DATA FILES-Sean McKenna is even topping himself with new mods to Microsoft BASIC. He now has KFILES, which is a data file handling system for BASIC designed to operate with cassette mass storage. KFILES will handle up to 8 files at one time with variable length buffers. Complete source listing and sufficient documentation are included. Contact Sean for more details at 64 Fairview Ave, Piedmont CA 94610.

Bet you can't top that, Sean!

PERRY PERIPHERALS has announced a package of information which will enable KIMSI/S100 users to use the HDE mini-floppy system. The info package sells for $15.00. Perry Peripherals is also a dealer for the HDE mini-floppy system hardware and software. Check with them for more details. Perry Peripherals, P.O.B. 924, Miller Place, NY 11764. Phone 516-744-6462.

COMPUTE MAGAZINE. The first issue of this mag really impressed me with its size and professional appearance. 104 glossy pages with good graphics, excellent layout and interesting content. Most of the information presented was PET oriented, but there were sections devoted to APPLE, ATARI, AIM, and OSIs C1P. Since most of the material came from the now defunct PET GAZETTE, that's not surprising. Now that 6502 USER NOTES has merged with COMPUTE, the single-board computer will be better represented.

COMPUTE also has a very revolutionary subscription option. They call it their "third level of domestic distribution". Besides the normal dealer distribution mail-order and subscription channels, COMPUTE offers a method whereby a subscriber can pick up his issue of COMPUTE at a nearby computer store (assuming its a COMPUTE dealer) for a reduced subscription rate. This saves money for everyone and promotes more traffic at the local dealer (your local dealer will like this also).

For more information on COMPUTE Magazine, contact:

COMPUTE
POB 5119
Greensboro NC 27403

6502 FUTURE (?)

At this time, it seems appropriate to say a few words about what the future looks like for the 6502 family devices-especially in light of Rockwell's move to second source the 68000 and Synerteks' apparent inactivity concerning their proposed 6516 (psuedo-16 bit 6502).

The 6500 series is not dead!!! It may be moving in a slightly different direction than some of the other upward-expanding 8-bit chips but it is not lying dormant!

Synertek and Rockwell are producing (or will be producing) new family devices such as the 6551 ACIA, the 6545 CRT controller, a floppy disk controller, bubble memory controller and a display controller (plus a few more besides). Rockwell has just finished a macro-assembler with relocating-linking loader for their System 65 and is pushing hard with the 6500/1 single chipper. They're planning to introduce a version of the 6500/1 with a piggyback EPROM socket for low volume and/or prototype applications.

Does that sound like a dead product line to you? It doesn't to me either.

Actually, I can't see any end to the need for 8-bit machines-especially clean machines like the 6500. Even if 16 bit super-micros (like the 68000) become the rage, 8-bit systems will still be the perfect solution for applications such as I/O processors, small controllers and the like.

So cheer up!!!!!!!!!!

software feature:

MATCH THIS

Gino F. Silvestri
Engineering Division
Loral Electronic Systems
999 Central Park Avenue
Yonkers, NY 10704

TRY TO MEMORIZE KIM's RANDOM TONE/LIGHT PATTERN-"BONUS" POINTS ARE GIVEN FOR REACHING "MILESTONES". AN INTERACTIVE GAME FOR A "NAKED" KIM-1.

This game requires a speaker/amplifier connection to the KIM-1 Application connector PAO port as shown on page 57 of the KIM-1 Users's Manual.

The game initializes page 0 locations by itself, and uses page 0 as a storage register for the game's moves. The program starts from "GO" at 0200 Hex, and occupies memory through 036D Hex.

When the "GO" button is pressed at 0200, a randomly chosen number (either a "0", "1", "2" or "3"), will appear in the KIM-1 display. The number will be positioned corresponding to the bottom four (0,1,2,3,)keys of the KIM keyboard. A tone related to the number displayed will come from the speaker.

The tone/character will appear briefly and stop-KIM awaits your response. Hit the key that matches the displayed character. If you hit the correct key, the same tone/number will be generated. The display will then light showing "b6C0 00", and the right digit will increment to display "b6C0 01" as you watch-this indicates that you've matched one step so far. KIM will now go back and play the first character, and then will add another at random-it may be the same as the preceding one-just play the keys as KIM directs.

When you successfully complete a sequence, the display will show the score you've reached. If you should strike an incorrect key during the sequence, KIM will immediately show an "E" at the display's left and sound a low "BUZZ" through the loudspeaker--the "Bonus Counter" score at the left (next to the "b") will be decremented by one-this means you have one less chance to continue the game (you started with 6 chances), and KIM will then go back to the beginning and replay the sequence to the point you had reached (your highest score at this point) before you made an error. KIM will now wait for your response to continue the game. The program will wait forever at this point-so there's no rush to go on. You may even press "GO" at this point to give up the whole game and restart from scratch if you like.

Continue play as KIM dictates, and you'll eventually repeat up to 6, 15 or 25 tone sequences These values are "Bonus Milestones" and you will get 1 extra "Bonus Point" in the Bonus Register for reaching each of these scores. A Bonus point represents one extra chance to continue the game for your highest score.

Should you make too many errors, the Bonus Counter will run out of chances. Just as the last "1" disappears from the Bonus display, an "L" will appear in the middle of the display, and you'll hear a low "raspberry" BUZZ tone from the speaker-this will alternate with a display of the highest score you reached before losing. KIM will keep buzzing and flashing like this forever (ignoring all other keypresses) until you press the "GO" button for a moment-this will restart the game from the very start-from scratch ("b6C0 00").

This game has no upper limit, although its score counter will roll over from 99 to 00 points, data will still be added to page zero memory. However, I don't believe anyone will have problems caused by getting that far. (The first person who does, can write a patch to add the "1" in front of the 001)

GOOD LUCK!!

"MATCH THIS"

PROGRAM LISTING FOR KIM-1

GINO F. SILVESTRI 12 FEBRUARY 1979

LABEL	ADDRESS A1 A2	DATA FIELD D1 D2 D3	OPCODE	FUNCTION	DESCRIPTION
START	02 00	D8	CLD		CLEAR DECIMAL MODE.
	01	A2 07	LDX#	NO. OF WORDS TO MOVE	INITIALIZE PAGE
MOVE	02 03	BD 55 03	LDAabs+X	FROM INITIAL DATA TO	ZERO FIELDS.
	06	95 D5	STAz+X	PAGE 0 FIELD STARTING WITH D5.	
	08	CA	DEX	NEXT ITEM TO MOVE.	
	09	10 F8	BPL	to MOVE UNTIL DONE.	
	0B	AD 04 17	LDAabs	from TIMER (KIM'S + 1) for RANDOM NUMBER.	
	0E	29 03	AND#	AND with 03 to MASK (STRIP to 0-3).	
	02 10	85 00	STAz	Put RANDOM NUMBER in 0000. (First move).	
	02 12	20 4A 03	JSR	"DELAY" Wait ¼ second.	
PLAY1	02 15	20 0E 03	JSR	"SOUNDS" Play tone/light display once.	
PLAY2	02 18	A5 DC	LDAz	check MODE reg for 1="TEST", 0="PLAY".	
	1A	D0 16	BNE	to "TEST2" if MODE="TEST".	
	1C	A5 D6	LDAz	get SEQUENCE COUNTER value.	
	1E	C5 D7	CMPz	compare to STEP COUNTER value, and go	
	02 20	F0 0C	BEQ	to "TEST1" if equal.	
	22	E6 D6	INCz	increment SEQUENCE COUNTER for next move.	
	24	A9 00	LDA#	zero MODE to "PLAY" mode,	
	26	85 DC	STAz	so "PLAY" can continue.	
	28	20 4A 03	JSR	"DELAY" Wait ¼ second.	
	2B	38	SEC	set carry for "branch always"	
	02 2C	B0 E7	BCS	to "PLAY1", to continue.	
TEST1	02 2E	A9 00	LDA#	zero SEQUENCE COUNTER to	
	02 30	85 D6	STAz	begin "TEST".	
TEST2	02 32	A9 01	LDA#	set MODE to "TEST".. ("TEST" = 1)	
	34	85 DC	STAz	store "1" in 'MODE.	
KEYIN	02 36	A9 00	LDA#	ready and clear DDR (Data Direction Register)	
	38	8D 41 17	STAabs	for safe "GETKEY" usage.	
	3B	AD 04 17	LDAabs	from KIM TIMER + 1 for RANDOM NUMBER and	
	3E	29 03	AND#	AND with 03 to MASK (STRIP to 0-3)	
	02 40	85 DD	STAz	store in RANDOM NUMBER for future use.	
	42	20 6A 1F	JSR	"GETKEY" KIM subroutine- What key is pressed?	
	45	C9 15	CMP#	if it's 15, it's NO KEY PRESSED, so it's back	

```
                47  F0 ED       BEQ     to "KEYIN" until a key is pressed.
                49  C9 13       CMP#    if it's 13, it's the GO key, so if it is- go
                4B  F0 B3       BEQ     to "START"-someone didn't like the game so far.
                4D  A6 D6       LDXz    get SEQUENCE COUNTER value for next instruction.
                4F  D5 00       CMPz+X  is the right key pressed? (0,1,2 or 3?), then go
           02 51  F0 33         BEQ     to "INCREMENT" to up the score.
                53  18         CLC     clear carry for illegal key check- if key value
                54  69 FC       ADC#    is added to FC, it'll cause a CARRY if over 3
           02 56  B0 DE         BCS     to "KEYIN" 'cause we'll ignore keys over 3.
                                        FALL THROUGH to "ERROR" if all above conditions
                                        are not met-therefore it must be the wrong key.
ERROR      02 58  A9 00         LDA#    zero LOOP STATUS for first pass showing
                5A  85 DE       STAz    bonus and score counters before loss of point.
                5C  A9 F9       LDA#    "E" character for display.
                5E  8D 40 17    STAabs  put in CHARACTER (PBD register).
           02 61  A9 09         LDA#    "E" will show up in leftmost position.
                63  8D 42 17    STAabs  put in POSITION register.
                66  A0 04       LDY#    "ERROR" tone value for "TONE" subroutine.
                68  20 1E 03    JSR     "TONE" - Sound "ERROR" tone- LOW "BUZZ".
SHOWLOSS        6B  20 F4 02     JSR     "SCORDIS" - Show bonus and score values.
                6E  A5 DE       LDAz    check LOOP STATUS to repeat or exit-
           02 70  D0 08         BNE     to ERREND to exit if second pass finished.
                72  C6 D5       DECz    decrement BONUS COUNTER 'cause you goofed!
                74  F0 62       BEQ     BONUS now "0"? too bad-go to "LOSE" subroutine.
                76  E6 DE       INCz    LOOP STATUS to "1"-don't decrement any more.
                78  D0 F1       BNE     to SHOWLOSS to display decremented bonus.
ERREND          7A  20 4A 03    JSR     "DELAY" Wait ½ second.
                7D  A9 00       LDA#    zero for:
                7F  85 D6       STAz    SEQUENCE COUNTER to start play from beginning,
           02 81  85 DC         STAz    MODE to "PLAY" for repeat of sequence.
           02 83  4C 15 02      JMP     "PLAY1"to remind you of sequence.

INCREMENT  02 86  20 0E 03      JSR     "SOUNDIS" -play for valid keypress.
KEYDOWN         89  20 40 1F    JSR     "KEYDOWN" KIM subroutine-wait for key release
                8C  D0 FB       BNE     to KEYDOWN until key is released-avoid errors.
                8E  A5 D7       LDAz    get STEP COUNTER value (highest step reached)
           02 90  C5 D6         CMPz    equal to SEQUENCE COUNTER? then go on
                92  F0 05       BEQ     to INCEND- (don't play any more-show score).
                94  E6 D6       INCz    well then, go on playing.
                96  4C 18 02    JMP     "PLAY2" to continue (but not from 0).
INCEND          99  E6 D7       INCz    increment STEP COUNTER to record progress.
                9B  A9 00       LDA#    zero LOOP STATUS for first score display
                9D  85 DE       STAz    to show increment of score in DECIMAL.
                9F  20 F4 02    JSR     "SCORDIS" to show bonus and score.
           02 A2  A5 DE         LDAz    check LOOP STATUS if one INCREMENT was done.
                A4  D0 10       BNE     to ONWARDS if it was, otherwise,
                A6  F8          SED     set DECIMAL mode for decimal score increment.
                A7  18          CLC     clear carry so decimal mode adds properly.
                A8  A9 01       LDA#    start with "01" in accumulator, and
                AA  65 D8       ADCz    add this to score in DECIMAL SCORE COUNTER (in acc)
                AC  85 D8       STAz    put result into DECIMAL SCORE COUNTER, and
                AE  D8          CLD     we've now finished a decimal increment.
                AF  A9 01       LDA#    make LOOP STATUS "1" so increment is not
           02 B1  85 DE         STAz    repeated again this time.
                B3  20 F4 02    JSR     "SCORDIS" to show bonus,score.


ONWARDS    02 B6  A2 02         LDX#    ready to test for 3 BONUS MILESTONES
BONUCHEK        B8  B5 D9       LDAz+X  start by checking DB, then DA, D9-
                BA  C5 D8       CMPz    does DECIMAL SCORE COUNTER equal any of these?
                BC  F0 05       BEQ     to BONUMET if one matches, continue checking
                BE  CA          DEX     by trying against next BONUS MILESTONE.
                BF  10 F7       BPL     to BONUCHEK if all milestones aren't tested.
           02 C1  30 06         BMI     to EXITINC since all milestones are tested.
BONUMET         C3  A9 FF       LDA#    if a milestone is reached, make it impossi-
                C5  95 D9       STAz+X  ible to match again this game.
                C7  E6 D5       INCz    increment BONUS COUNTER for MILESTONE was met.
EXITINC         C9  A6 D7       LDXz    ready to store RANDOM NUMBER in its new spot.
                CB  A5 DD       LDAz    get RANDOM NUMBER that was generated before,
                CD  95 00       STAz+X  and store in new page zero location.
                CF  A9 00       LDA#    ready to go back to play mode to continue.
           02 D1  85 DC         STAz    MODE to "PLAY" (MODE=0)
                D3  85 D6       STAz    SEQUENCE COUNTER to "0" to play from beginning.
           02 D5  4C 15 02      JMP     "PLAY1"  Play the stored sequence from pg. 0.

LOSE       02 D8  A9 B8         LDA#    "L" character for "LOSE" display.
                DA  8D 40 17    STAabs  in 'CHARACTER register.
                DD  A9 0F       LDA#    fourth position in display.
                DF  8D 42 17    STAabs  in POSITION register.
           02 E2  A0 05         LDY#    "LOSE" tone value (Low BUZZ).
                E4  20 1E 03    JSR     "TONE" - sound for loss.
                E7  20 F4 02    JSR     "SCORDIS" - show score reached before loss.
                EA  20 6A 1F    JSR     "GETKEY" (KIM subroutine) only way out of this-
                ED  C9 13       CMP#    if key is "GO" key-we'll start over again,
                EF  D0 E7       BNE     to LOSE, to stay for good otherwise.
           02 F1  4C 00 02      JMP     to START to begin from scratch.
```

```
SCORDIS   02 F4  A5 D5      LDAz    get BONUS COUNTER value for display.
          F6  09 B0        ORA     put a "B" in front of value (could be 1-9).
          F8  85 FB        STAz    put "Bx" in SCANDS page zero register-(LEFT).
          FA  A9 C0        LDA#    "C0" for center display for COunt.
          FC  85 FA        STAz    put in SCANDS page zero register-(CENTER).
          FE  A5 D8        LDAz    get value of DECIMAL SCORE COUNTER.
       03 00  85 F9        STAz    put in SCANDS page zero register-(RIGHT)
          02  A9 FF        LDA#    starting value for SCANDS counter.
          04  85 D3        STAz    load SCANDS counter for display time.
SCANDS    06  20 1F 1F     JSR     KIM SCANDS subroutine for display.
          09  C6 D3        DECz    decrement SCANDS counter (display time).
          0B  D0 F9        BNE     to SCANDS if display time not up yet.
       03 0D  60           RTS     return from SCORDIS subroutine.

SOUNDIS   03 0E  A6 D6      LDXz    get SEQUENCE COUNTER VALUE-where are we?
       03 10  B4 00        LDYz+X  get data for this routine from page zero.
          12  B9 E7 1F     LDAabs+Y convert data to character using KIM rom table.
          15  8D 40 17     STAabs  store data in CHARACTER (char= "0","1","2", or "3".)
          18  B9 5D 03     LDAabs+Y use Y offset in table to find POSITION.
          1B  8D 42 17     STAabs  in 1742-POSITION register for display.
TONE      03 1E  BE 67 03  LDXabs+Y get TONE TIME for this item from lookup table.
       03 21  86 D4        STXz    put this  value in page zero counter.
          23  A9 7F        LDA#    ready to open port of B Data Direction Register.
          25  8D 41 17     STAabs  open port for display of character.
          28  A9 01        LDA#    initial data for PA0 port for speaker.

       03 2A  8D 01 17     STAabs  open PA0 port for speaker.
REPEAT    2D  8D 00 17     STAabs  send data out to speaker, "on" or "off".
       03 30  BE 61 03     LDXabs+Y get TONE data from lookup table.
          33  8E 06 17     STXabs  start KIM timer (* 64) (how long on or off).
BIT1      36  2C 07 17     BITabs  time up yet?
          39  10 FB        BPL     to BIT1 if not done, otherwise go on to
          3B  49 01        EOR     exclusive OR accum. with 01 to flip spkr. bit.
          3D  C6 D4        DECz    decrement TONE TIME register.
          3F  D0 EC        BNE     to REPEAT to send flipped bit to speaker.
       03 41  A9 00        LDA#    zero so as to end SOUNDIS routine by
          43  8D 01 17     STAabs  closing the speaker port, (no DC to speaker),
          46  8D 40 17     STAabs  and closing the display port.
       03 49  60           RTS     SOUNDIS done-back to where you came from.

DELAY     03 4A  A9 FF      LDA#    ready for maximum delay time (250 mS).
          4C  8D 07 17     STAabs  start KIM timer (* 1024).
 BIT2     4F  2C 07 17     BITabs  check for time up.
       03 52  10 FB        BPL     done? back if not, otherwise go on. (back to BIT2)
       03 54  60           RTS     back to where you came from.
                 INITIAL DATA FIELD FOR START ROUTINE
          03 55  06        DATA 1  BONUS COUNTER starting value for 00D5.
             56  00        DATA 2  SEQUENCE COUNTER starts at "00"-for 00D6.
             57  00        DATA 3  STEP COUNTER starts at "00"-for 00D7.
             58  00        DATA 4  DECIMAL SCORE COUNTER to "00" for 00D8.
             59  06        DATA 5  MILESTONE 1-Get past "06" and get a BONUS POINT.
             5A  15        DATA 6  MILESTONE 2-Pass"15" and get another point.(DA)
             5B  25        DATA 7  MILESTONE 3-Pass "25" and get yet another.(00DB)
          03 5C  00        DATA 8  MODE starts in "PLAY" ("00") mode.(00DC)
                   LOOKUP TABLE VALUES FOR "SOUNDIS" ROUTINE
POS DATA  03 5D  09        DATA 9  FIRST (leftmost) character position in display.
             5E  0B        DATA 10 SECOND character position.
             5F  0D        DATA 11 THIRD character position.
          03 60  0F        DATA 12 FOURTH character position.
TONE DATA 03 61  88        DATA 13 (62 Hz) TONE for character "0".
             62  35        DATA 14 (150 Hz) TONE for character "1".
             63  18        DATA 15 (325 Hz) TONE for character "2".
             64  11        DATA 16 (448 Hz) TONE for character "3".
             65  B0        DATA 17 "ERROR" TONE for "E" character.
             66  C0        DATA 18 "LOSE" TONE for "L" character.
TIME DATA 03 67  20        DATA 19 (230 mS) TIME value for "0" tone.
             68  50        DATA 20 (230 mS) TIME value for "1" tone.
             69  B0        DATA 21 (230 mS) TIME value for "2" tone.
             6A  FF        DATA 22 (230 mS) TIME value for "3" tone.
             6B  80        DATA 23 "ERROR" tone time-3 seconds.
             6C  55        DATA 24 "LOSE" tone time-2 seconds.
END       03 6D -----LAST ADDRESS-----------------------------------------------
```

OPCODE SYMBOL REMINDER:
- #= IMMEDIATE ADDRESSING MODE.
- z= ZERO PAGE ADDRESSING MODE.
- abs= ABSOLUTE ADDRESSING MODE.
- +Y,+X= MODE INDEXED BY X OR Y REGISTERS.

LANGUAGE LAB

basic

HOW TO TRANSFER BASIC PROGRAMS FROM PET TO KIM

Rush Shijanowski
Eric C. Rehnke

If you have Microsoft BASIC running on your KIM, you are already aware of the fact that there aren't many BASIC programs available on KIM cassette! On the other hand, I've managed to collect a fairly large number of programs for my Pet. Since the KIM has floppies and the Pet only has cassette for mass storage, it seemed a natural to transfer the BASIC programs from Pet to KIM.

Since typing the programs into KIM was out of the question (I'm lazy), I searched around for a way to make the two computers do all the work (that's why we have computers, right?)

It wasn't until I cam across a program written by Rush Shijanowswi that the end of my quest came into view.

Rush programmed his KIM to receive data from Pets' IEEE port and list it out on KIM's printer. He took advantage of the fact that Pet can list a program in ASCII to its' IEEE port.

I modified his program to also save the ASCII text in a buffer for later recovery by KIM BASIC. This would be done by writing a new input routine for BASIC which would get its' input from a text buffer instead of the terminal.

Of course, the ultimate solution would entail further modification to the 'IEEE to KIM test program' to permit IT to be the input routine for KIM BASIC. This would simplify the number of work steps but I'm not sure how KIM BASIC would interpret commands which are not in its reperatory, such as OPEN, CLOSE etc.

This same technique for getting a computer to LIST a program to some output device can be used to "recover" BASIC programs from other machines such as TRS-80 and probably Apple. (I'm sure about TRS-80 because I saw an article in Kilobaud on how to hook up a printer and list out to it. Nothing says that the printer can't be a hungry BIG KIM!)

Here are the commands to make your PET list out to the IEEE port. To open the bus use, OPEN 4,4
CMD 4

Then to list a program type LIST

Well, that's a start. You can take it from here.

Eric

PET IEEE PORT Pin	Signal	KIM-1 Applications Connector	Pin	
1	DIO1	PA0	14	
2	DIO2	PA1	4	
3	DIO3	PA2	3	
4	DIO4	PA3	2	
A	DIO5	PA4	5	Data
B	DIO6	PA5	6	
C	DIO7	PA6	7	
D	DIO8	PA7	8	
11	ATN	PB5	16	
5	EOI	PB4	13	Management
6	DAV	PB7	15	
7	NRFD	PB1	10	Handshake
8	NDAC	PB0	9	
	GROUND	GROUND		

```
01-0010  2000              ;IEEE TO KIM TEST PROGRAM
01-0012  2000              ;WRITTEN BY RUSH SHIJANOWSKI
01-0013  2000              ;MODIFIED BY ERIC C. REHNKE
01-0020  2000
01-0030  2000              PADD   =$1701
01-0040  2000              PBDD   =$1703
01-0050  2000              PAD    =$1700
01-0060  2000              PBD    =$1702
01-0070  2000              BUFFER =$2100
01-0080  2000
01-0090  2000              POINTL =$0000
01-0100  2000              POINTH =$0001
01-0110  2000              OUTCH  =$1EA0
01-0120  2000              CRLF   =$1E2F
01-0130  2000
01-0140  2000                     *=$2000
01-0150  2000
01-0160  2000  A9 00       START  LDA #0          ;SETUP I/O
01-0170  2002  8D 01 17           STA PADD        ;ON FOR KIM
01-0180  2005  A9 03              LDA #$3         ;TO RECEIVE.
01-0190  2007  8D 03 17           STA PBDD
01-0200  200A  A9 00              LDA #<BUFFER    ;SETUP BUFFER
01-0210  200C  85 00              STA POINTL      ;IN KIM
01-0220  200E  A9 21              LDA #>BUFFER
01-0230  2010  85 01              STA POINTH
01-0240  2012  A9 02       LOOP   LDA #2          ;NRFD HIGH, NDAC LOW
01-0250  2014  8D 02 17           STA PBD
01-0260  2017  AD 02 17    DAVOFF LDA PBD         ;WAIT FOR DAV
01-0270  201A  30 FB              BMI DAVOFF
01-0280  201C  CE 02 17           DEC PBD         ;NRFD LOW, NDAC HIGH
01-0290  201F  29 20              AND #$20        ;IGNORE BYTES WITH ATN
01-0300  2021  F0 20              BEQ DAVON
01-0310  2023  AD 00 17           LDA PAD         ;GET DATA
01-0320  2026  49 FF              EOR #$FF        ;INVERT IT
01-0330  2028  A0 00              LDY #0
01-0335  202A  C9 0A              CMP #$0A        ;IS IT A LINE FEED?
01-0336  202C  F0 15              BEQ DAVON       ;IGNORE IT
```

```
01-0340  202E  91 00          STA (POINTL),Y   ;STORE IT AWAY
01-0350  2030  E6 00          INC POINTL
01-0360  2032  D0 02          BNE OUT
01-0370  2034  E6 01          INC POINTH
01-0400  2036  C9 0D    OUT   CMP #$0D         ;IS IT A CARRIAGE RETURN?
01-0410  2038  D0 06          BNE PRINT        ;NO, THEN SKIP CRLF
01-0420  203A  20 2F 1E       JSR CRLF
01-0430  203D  4C 43 20       JMP DAVON
01-0440  2040  20 A0 1E PRINT JSR OUTCH
01-0450  2043  2C 02 17 DAVON BIT PBD          ;WAIT FOR NOT DAV
01-0460  2046  10 FB          BPL DAVON
01-0470  2048  30 C8          BMI LOOP
01-0480  204A                 .END
```

BASIC CASSETTE I/O MODS

Glen Deas
PO Box 73
Ruston, La 71270

I am sending along my versions of CSAVE & CLOAD for the Johnson Computer Company 8.5 K BASIC. I noted wzth interest Don Latham's comments, Vol 12, on the system hanging up on a bad load. My read routine causes a return to command mode after printing????,meaning a load error occurred. Seems to work OK; nothing will list out after a bad load, but you could probably find the error location by poking around wzth the pointers (120-123 decimal) to list it out. I have yet to get any load errors except those I induced to test the routine. I am using an el cheapo General Electric cassette model 335013A (Note: it is the only one I've found around here that works for recorder-recorder duplicating, even Hypertape) that works FINE (in fact, better than most of the more expensive ones we have here).

For those who may not know, you can tack on other programs (subroutines, data stat, etc.) like so:

```
PRINT    PEEK (120),  PEEK (121)
   XXX   YYY

PRINT    PEEK (122),  PEEK (123)
   ZZZ   AAA
```

```
2000
2000            ;*********************
2000            ;*KIM-1  8K BASIC   *
2000            ;*CASSETTE   SAVE   *
2000            ;*SUBROUTINE        *
2000            ;*                  *
2000            ;* A MODIFIED       *
2000            ;*VERSION OF HYPER  *
2000            ;*TAPE (JIM BUTTER  *
2000            ;*FIELD)     GED    *
2000            ;*********************
2000
2000            ;PATCHES: $275C 20 00 02
2000
2000            VEB     =$17EC
2000            SBD     =$1742
2000            SAL     =$17F5
2000            SAH     =$17F6
2000            EAL     =$17F7
2000            EAH     =$17F8
2000            PBDD    =$1743
2000            CLKONE  =$1744
2000            CLKRDI  =$1747
2000            ID      =$17F9
2000            CHKT    =$194C
2000            INCVEB  =$19EA
2000            CHKL    =$17E7
2000            CHKH    =$17E8
2000            INITA   =$1E8C
2000            INTVEB  =$1932
2000
2000            ;ZERO PAGE
2000
2000            TIC     =$00F1
2000            COUNT   =$00F2
2000            TRIB    =$00F3
2000            GANG    =$00F5
2000
```

ZZZ is the low order byte (dec. value) of the end pointer. Subtract 2 from this value (call it BBB); if the result is negative, subtract 1 from AAA. (Call it CCC) then

```
POKE 120, BBB : POKE 121, CCC
LOAD
```

Then restore 120 & 121 to their original values

```
POKE 120, XXX : POKE 121, YYY
```

Caution: The additional lines should have line numbers greater than the last statement of the original program.

Hope you can use some of this.

When you record the PATCHED VERSION of BASIC, make sure you record location 4260 (null char)--basic bombs out without it!

```
17F5 00
17F6 20
17F7 61
17F8 42
```

```
2000
2000                    *=$0200
0200
0200  A9 AD       CSAVE  LDA #$AD          ;LDA INSTR
0202  8D EC 17           STA VEB
0205  20 32 19           JSR INTVEB SET UP SUB
0208
0208  A9 27              LDA #$27
020A  85 F5              STA GANG FLOP FLAG
020C  A9 BF              LDA #$BF
020E  8D 43 17           STA PBDD DIR REG
0211
0211  A2 FA              LDX #$FA SEND 250
0213  A9 16              LDA #$16 SYNC CHAR...
0215  20 61 02           JSR HIC
0218  A9 2A              LDA #'* START OF FILE
021A  20 88 02           JSR OUTCHT
021D
021D  AD F9 17           LDA ID
0220  20 70 02           JSR OUTBT PGM ID
0223  AD F5 17           LDA SAL AND START ADR
0226  20 6D 02           JSR OUTBTC SEND AND CHKSUM
0229  AD F6 17           LDA SAH
022C  20 6D 02           JSR OUTBTC
022F
022F  20 EC 17     DATA  JSR VEB
0232  20 6D 02           JSR OUTBTC SEND BYTE
0235  20 EA 19           JSR INCVEB MOVE TO NEXT
0238  AD ED 17           LDA VEB+1
023B  CD F7 17           CMP EAL LAST BYTE?
023E  AD EE 17           LDA VEB+2
0241  ED F8 17           SBC EAH
0244  90 E9              BCC DATA NO-REPEAT
0246
0246  A9 2F              LDA #'/ YES-END OF FILE
0248  20 88 02           JSR OUTCHT ASCII VALUE
024B  AD E7 17           LDA CHKL SEND CHKSUM
024E  20 70 02           JSR OUTBT
0251  AD E8 17           LDA CHKH
```

```
0254  20 70 02    EXIT   JSR OUTBT
0257  A2 02              LDX #$02
0259  A9 04              LDA #$04  2 EOT CHARS
025B  20 61 02           JSR HIC
025E
025E  4C BE 02           JMP RETURN
0261
0261                  ; SUBROUTINES
0261
0261  86 F1        HIC    STX TIC
0263  48          HICK   PHA
0264  20 88 02           JSR OUTCHT
0267  68                 PLA
0268  C6 F1              DEC TIC
026A  D0 F7              BNE HICK
026C  60                 RTS
026D
026D  20 4C 19    OUTBTC JSR CHKT
0270  48          OUTBT  PHA
0271  4A                 LSR A GET LEFT NIBBLE
0272  4A                 LSR A
0273  4A                 LSR A
0274  4A                 LSR A
0275  20 7D 02           JSR HEXOUT SEND IT
0278  68                 PLA NOW THE RIGHT
0279  20 7D 02           JSR HEXOUT
027C  60                 RTS

027D  29 0F        HEXOUT AND #$0F CLEAR LEFT BITS
027F  C9 0A              CMP #$0A >10?
0281  18                 CLC
0282  30 02              BMI HEXA
0284  69 07              ADC #$07 ADD 37 IF ALPHA
0286  69 30        HEXA   ADC #$30 30 IF NUM.
0288
0288  A0 07        OUTCHT LDY #$07 FOR 8 BITS
028A  84 F2              STY COUNT
028C  A0 02        TRY    LDY #$02 SEND 3 UNITS
028E  84 F3              STY TRIB OF 3600 HZ
0290  BE C3 02     ZON    LDX NPUL,Y
0293  48                 PHA
0294  2C 47 17     ZONA   BIT CLKRDI WAIT FOR
0297  10 FB              BPL ZONA TIMEOUT
0299  B9 C4 02           LDA TIMG,Y
029C  8D 44 17           STA CLKONE
029F  A5 F5              LDA GANG
02A1  49 80              EOR #$80 FLIP 1-0-1-0..
02A3  8D 42 17           STA SBD
02A6  85 F5              STA GANG
02A8  CA                 DEX ALL CYCLES SENT?
02A9  D0 E9              BNE ZONA NO,GO BACK
02AB  68                 PLA YES,PULL CHAR
02AC  C6 F3              DEC TRIB ONE LESS
02AE  F0 05              BEQ SETZ BR. IF LAST
02B0  30 07              BMI ROUT BR IF NO MORE
02B2  4A                 LSR A
02B3  90 DB              BCC ZON IF ITS A ONE
02B5  A0 00        SETZ   LDY #$00 THEN 2400 HZ
02B7  F0 D7              BEQ ZON FORCED BRANCH
02B9  C6 F2        ROUT   DEC COUNT ONE LESS BIT
02BB  10 CF              BPL TRY ANY MORE? GO BACK
02BD  60                 RTS
02BE
02BE  20 8C 1E     RETURN JSR INITA RESET PORTS
02C1  58                 CLI ACCEPT INTER. NOW
02C2  60                 RTS AND BACK TO BASIC
02C3
02C3
02C3                  ; FREQ/DENSITY
02C3  02           NPUL   .BYTE $02        ;2 PULSES
02C4  C3           TIMG   .BYTE $C3        ;2400 HZ
02C5  03           TRW    .BYTE $03        ;3 PULSES
02C6  7E           GED    .BYTE $7E        ;3600 HZ
02C7
02C7                      .END
```

```
2000              ;*********************
2000              ;* KIM-1  8K BASIC   *
2000              ;* CASSETTE   LOAD   *
2000              ;* ROUTINE           *
2000              ;*                   *
2000              ;*    GLEN DEAS      *
2000              ;*    PO BOX 73      *
2000              ;*    RUSTON, LA.    *
2000              ;*       71270       *
2000              ;*********************
2000              ;PATCHES TO BASIC:
2000              ;$2763 - 76
2000              ;$2769 - 4C 4F 41 44
2000              ;        45 44 0D 0A
2000              ;        4F 4B 0D 0A
2000              ;        00 53 41 56
2000              ;        45 44 0D 0A
2000              ;        00
2000              ;LOADED OK : SAVED
2000              ;$27A3 - 4C 00 03
2000              ;  JMP TO LOAD ROUTINE
2000              ;
2000              ;ZERO PAGE USAGE: ONLY
2000              ;    NORMAL KIM USAGE
2000              ;ID FUNCTIONS RETAINED
2000              ;FOR POSSIBLE FUTURE
2000              ;USE
2000
2000
2000              VEB     =$17EC
2000              SBD     =$1742
2000              SAVX    =$17E9
2000              RDCHT   =$1A24
2000              RDBIT   =$1A41
2000              RDBYT   =$19F3
2000              ID      =$17F9
2000              CHKT    =$194C
2000              PACKT   =$1A00
2000              INCVEB  =$19EA
2000              CHKL    =$17E7
2000              CHKH    =$17E8
2000              INITA   =$1E8C
2000              OUTCH   =$1EA0
2000              INTVEB  =$1932
2000
2000              GUDLOD  =$27A6
2000              BADLOD  =$2523
2000
2000
2000                CLOAD  *=$0300
0300  A9 8D       LOADT  LDA #$8D STA ABS. INSTR.
0302  8D EC 17           STA VEB
0305  20 32 19           JSR INTVEB
0308  A9 4C              LDA #$4C
030A  8D EF 17           STA VEB+3 RET.BY JUMP
030D  AD C9 03           LDA TAB
0310  8D F0 17           STA VEB+4
0313  AD CA 03           LDA TAB+1
0316  8D F1 17           STA VEB+5
0319                  ;
0319  A9 07              LDA #$07 RESET PB5=0
031B  8D 42 17           STA SBD
031E                  ;
031E                  ;
031E  A9 FF        SYNC   LDA #$FF
0320  8D E9 17           STA SAVX
0323                  ;
0323  20 41 1A     SYNCA  JSR RDBIT
0326  4E E9 17           LSR SAVX
0329  0D E9 17           ORA SAVX
032C  8D E9 17           STA SAVX
032F  AD E9 17           LDA SAVX
0332  C9 16              CMP #$16 SYN CHAR.
0334  D0 ED              BNE SYNCA
0336                  ;
0336  A2 0A              LDX #$0A TEST FOR 10 SYN
0338  20 24 1A     SYNCB  JSR RDCHT
033B  C9 16              CMP #$16
033D  D0 DF              BNE SYNC IF NOT THEN AGN
033F  CA                 DEX
0340  D0 F6              BNE SYNCB
0342                  ;
0342                  ;
0342  20 24 1A     LOADTD JSR RDCHT
0345  C9 2A              CMP #'* START OF DATA?
0347  F0 06              BEQ LOADII
0349  C9 16              CMP #$16 IF NOT *
034B  D0 D1              BNE SYNC
034D  F0 F3              BEQ LOADTD
```

```
034F                    ;
034F  20 F3 19    LOADII JSR RDBYT
0352  CD F9 17           CMP ID RIGHT FILE?
0355  F0 0D              BEQ LOADTE
0357  AD F9 17           LDA ID
035A  C9 00              CMP #$00 DEFAULT MODE
035C  F0 06              BEQ LOADTE READ ANYWAY
035E  C9 FF              CMP #$FF
0360  F0 17              BEQ LOADTF IGNORE SA
0362  D0 9C              BNE LOADT

0364  20 F3 19    LOADTE JSR RDBYT GET SA
0367  20 4C 19           JSR CHKT
036A  8D ED 17           STA VEB+1
036D                    ;
036D  20 F3 19           JSR RDBYT
0370  20 4C 19           JSR CHKT
0373  8D EE 17           STA VEB+2
0376  4C 85 03           JMP LOADTG
0379                    ;
0379  20 F3 19    LOADTF JSR RDBYT GET SA
037C  20 4C 19           JSR CHKT BUT IGNORE
037F  20 F3 19           JSR RDBYT
0382  20 4C 19           JSR CHKT
0385                    ;
0385  A2 02       LOADTG LDX #$02 GET 2
0387  20 24 1A    LOADIC JSR RDCHT CHAR.
038A  C9 2F              CMP #'/ END OF FILE?
038C  F0 14              BEQ LOADTH
038E  20 00 1A           JSR PACKT
0391  D0 25              BNE LOADTI
0393  CA                DEX
0394  D0 F1              BNE LOADIC
0396                    ;
0396  20 4C 19           JSR CHKT
0399  4C EC 17           JMP VEB
039C  20 EA 19    LOADIB JSR INCVEB
039F  4C 85 03           JMP LOADTG
03A2                    ;
03A2  20 F3 19    LOADTH JSR RDBYT CHKSUM
03A5  CD E7 17           CMP CHKL
03A8  D0 0E              BNE LOADTI
03AA  20 F3 19           JSR RDBYT
03AD  CD E8 17           CMP CHKH
03B0  D0 06              BNE LOADTI
03B2
03B2  20 8C 1E           JSR INITA
03B5  4C A6 27           JMP GUDLOD
03B8                    ;
03B8  20 8C 1E    LOADTI JSR INITA RESET PORTS
03BB  D8                 CLD JUST IN CASE
03BC  A2 04              LDX #$04
03BE  A9 3F       HUH    LDA #'? ERRORS
03C0  20 A0 1E           JSR OUTCH
03C3  CA                 DEX
03C4  D0 F8              BNE HUH
03C6
03C6  4C 23 25           JMP BADLOD
03C9
03C9                    ;RETURN ADR FROM VEB
03C9
03C9  9C          TAB    .BYTE $9C,$03
03CA  03
03CB                     .END
```

LOAD MULTIPLE FILES IN BASIC

H J Schilling
DJ1XK

Normally, MICROSOFT BASIC for KIM-1 doesn't allow to load multiple files of source code. But there is a little trick to load more than one source file into memory, allowing use of prepared subroutines, data statements with tables or the RENUMBERING program (see 6502 USER NOTES # 10).

For loading a file, KIM-1 BASIC takes the "pointer to start of program" in $78, $79 as the start address for the loader in $17F5, $17F6. In $7A, $7B, however, the "pointer to start of array table" minus 3 is the end of the former loaded program, and you only have to transfer this address to $78, $79 before the second LOAD command. Remember that the addresses are in LO,HI order, and make the correct borrow when substracting the "3"! If you intend to load another file, you have to transfer the new address from $7A, $7B to $78, $79 again. After the last LOAD you must correct the start address as BASIC needs it for RUN etc.

Don't forget that the line numbers must be in ascending order, e.g. the separate files must have line numbers in different blocks with correct order!

Example:

```
NEW
OK
LOAD
LOADED
PRINT PEEK (120);PEEK(121);PEEK(122);PEEK(123)
 66   64   141   65
OK
POKE 120,138:POKE 121,65
OK
LOAD
LOADED
POKE 120,66:POKE 121,64
OK
```

tiny basic

Ben Doutre
621 Doyle Rd
Mont St-Hilaire Que
Canada J3H 1M3

Dear Eric,

First, let me say that 6502 User Notes is top quality and getting better with each issue. Keep up the good work.

I have been following the Tiny Basic items with particular interest and feel that Michael Day, Lew Edwards and William Clements are to be congradulated for their contributions in issues #13-15. The following comments may be of interest:

a) In Day's string mods, KIM owners who are using the TTY I/O routines GETCH and OUTCH will have problems, since these do not save the Y register. Rather than reassemble the code, you can set up a couple of buffer I/O routines as follows:

```
INPUT   JSR   GETCH      OUTPUT JSR OUTCH
        INY                      INY
        RTS                      RTS
```

and change your JMP vectors at $0206 and $0209 to wherever you tuck these routines in. There is also a pretty obvious typo at 0B82: 02 should be 20. These string features are really interesting to play with. (The BNE instruction at $0B7B in Tiny B must be changed to BEQ for this mod to work).

b) In Clements tape SAVE and LOAD mod, one item was omitted from the list of revised branches: at IL relative address 00DD, the "30E2" should be changed to "30F9". This mod also works great, although perosnally, I have reservations about adding IL workload (I seldom use "Let" expressions) for non-run-time extensions and prefer to use an input trap routine. But that is another story.

SUPERKIM

- 5 volt 3 amp, 12 volt .1 amp power supply (less AC transformer)
- Up to four bidirectional 8 bit in or out serial shift registers (1 6522 supplied)
- Up to 9 counter/times (3 supplied)
- Up to 4K bytes of 2114 static RAM (1K supplied)
- Up to 16K EPROM (2732) or 8K EPROM (2716)
- Up to 9 bidirectional 8 bit in/out ports (3 supplied, 2-6530, 1-6522)
- Up to 4 programmable tone generators (1 6522 supplied)
- 8 vectored, priority, latched interrupts (4 separate real time clocks possible)
- RS232 serial interface, TTY interface
- 3" x 10" prototype area
- KIM-1* audio tape interface, totally KIM-1 software compatible
- 11-1/2" x 11-1/2" double sided, solder masked, singleboard computer, fully socketed
- 200 gold wire wrap pins for easy connection to CPU buss and all in and out pins to wire wrap sockets installed in the prototype area
- 20 key Hex keypad with gold plated PC board, tactile feedback and separate injection molded keys (can be remotely mounted)
- 6, 7 segment LED's on separate piggy backed PC board (can be removed for remote mounting)

Here is a powerful microprocessor control system development tool and a complete real-time multitasking microcomputer in one package. There is no need to buy a power supply, motherboard, memory boards and separate I/O boards when your requirements may be satisfied by a SUPERKIM. You may only need a couple of wire-wrap sockets and a few LSI chips installed in the big 3" x 10" onboard prototype area to accomplish the required memory expansion and interface with the real world.

Some single chip interface devices available are: UARTS, 16 channel-8 bit analog to digital data acquisition systems, floppy disk controllers and dot matrix printer controllers. Furthermore, you will shortly be able to buy single 5 volt supply pseudo static 8K byte (that's right, you read it right, 8K x 8 bits) memory chips in a single 28 pin package. These chips use the same technology developed for the 64K bit dynamic RAMs now being manufactured by TI, MOTOROLA and others. Just five of these chips and four 2732 EPROMs in the sockets already supplied in the SUPERKIM will yield a fully populated SUPERKIM with 44K bytes of RAM, 16K bytes of EPROM with serial and parallel I/O ports, and enough room leftover in the prototype area for a LSI floppy disk controller chip. MOSTEK already has, on the market, a 2K byte version of this memory chip that is pin compatible with the 8K byte version; no need to rewire your sockets when the larger memories become available. Put in 14K now and upgrade later to 44 K.

If you started with a KIM-1, SYM-1 or AIM-65 and tried to expand it to the basic capabilities of the SUPERKIM, you would need a power supply ($60), a motherboard ($120), a prototype board ($30), a memory board ($120), and an I/O board ($120) for a total cost of from $620 in the case of the KIM-1 to $825 in the case of the AIM-65. You still would not have real time multitasking capabilities.

Multitasking is a situation where the microcomputer appears to be doing more than one job simultaneously. For example, the microcomputer could be sending data to a printer, accepting analog data from a 16-channel data acquisition system and presenting data to an operator monitoring a LCD or LED display, all the while keeping track of time.

Multitasking is accomplished on the SUPERKIM by use of vectored priority interrupts and a real time clock. This real time clock is implemented using one of the four onboard 6522 programmable tone generators.

The SUPERKIM, with its keyboard, display and ROM monitor, can be used as a system analyzer for troubleshooting hardware and software in-the-field or during the system development as an in circuit emulator. The monitor can stop the CPU at any point in the program, step through the program, change the contents of the systems' memory and CPU registers, and record the CPU's registers during a selected portion of the program. It offers one of the most powerful combinations of development and diagnostic tools available on the market today.

All of the above is unavailable on any other singleboard computer at any price.

* KIM-1 is a product of MOS technology

$395

microproducts

2107 ARTESIA BOULEVARD • REDONDO BEACH, CALIFORNIA 90278 • (213) 374-1673

I have developed a small (74 bytes) utility program which makes it pretty easy and straightforward to load machine-code routines. If you feel that your readers would be interested, the enclosed listing and example of use will make most of it clear, together with these additional comments.

My system is a KIM-1 with an additional 8K bytes of RAM, located at $2000 to $3FFF. My version of Tiny Basic is TB651T, V.1T, which loads at $2000 and extends to $28C6. Day's multiple statement per line mods are tucked into the remaining $2800 space, and the next 1K is allocated to utilities, like tape I/O (I use Lew Edwards' ZIPTAPE, the greatest thing to come along since sliced bread!), Selectric print routines, etc. User space is allocated starting at $2D00, but this can vary.

EZLOAD is an interface routine which scans the output stream looking for a unique prefix character. When it finds it, it then proceeds to convert each following pair of characters into a hex byte which is placed at the top (bottom?) of the Basic stack. Anyway, the bytes are shuffled along the stack, with the Basic stack pointer and variable "A" (an arbitrary choice) keeping up with the head of the code. The loading stops when a carriage return comes along, but may resume and stop several times. When the dust finally settles, the machine code is neatly arranged in execution order at the top of user space, with not a byte wasted, and with "A" all set to be used as the first parameter in a USR function call.

The machine code is written into REM statements, and will print in readable form when listed. It is, in fact, loaded by being LISTed, and is effectively wiped out by a warm start (the Basic stack pointer is reset) or by the execution of an END statement, which ends up doing a warm start for you. The best way to use a program with EZLOAD machine code is to do a command-mode END, list the program, then RUN it.

The code will not load when you are first typing it in, unless you have an I/O setup with external echo. You may be tempted to use the selected prefix character in a run-time PRINT "..." but this will clobber your stack when it is in use for other things. With some slight changes, though, this presents some intriguing possibilities. Obviously, the programs may be saved on tape, and later loaded with their machine-code still intact and usable. This is a considerable benefit.

EZLOAD was written with severe space constraints, consequently some niceties were left out, such as checking for stack over flow. In particular, it will not work as is unless some modifications are made to Tiny's memory grab code in the cold start areas. These are detailed below. Users with more bytes available might want to check for valid HEX code characters (KIM's PACKT will return with Zero bit set if valid, reset otherwise, assuming you enter with Y equal 0) and use the validity check to step over spaces and other readability aids. You could also use several of Tiny's variables to point to various code segments, or several different prefixes, etc etc.

The trouble with the cold start code, insofar as this program is concerned, is that it runs the top-of-user-space pointer ($0022-23) to the last real RAM location plus one. That plus one I didn't need! And contrary to what the Experimenter's Kit seems to say (top of page 6), the Basic stack pointer must be decremented before use, not after; these conditions presented severe problems in initializing EZLOAD, beyond resetting the load flag which is done by the first carriage return from a warm start. So that cute memory grab finally had to go!

In my version of TB, the cold start vector jump at $2000 points to $2085. The code from $2085 thru $20A9 initializes both the start and end of user space pointers ($0020-21 and $0022-23, respectively). The following code was substituted: (You should, of course, use your own start and end values):

```
2085 A9 00    COLDST LDA #$00
2087 85 20           STA $20
2089 A9 2D           LDA #$2D
208B 85 21           STA $21   ; user space start
                                 at $2D00
208D A9 FF           LDA #$FF
208F 85 22           STA $22
2091 A9 3F           LDA #$3F
2093 85 23           STA $23   ; user space end at
                                 $3FFF
2095 A0 00           LDY #$00 ; zero Y register
2097 4C AA 20        JMP $20AA ; for rest of init
       ......
20AA D8              CLD       ; existing code
20AB A5 20           LDA $20
         etc
```

In the following warm start code, the Basic stack pointer $0026-27 is made equal to top-of-user-space pointer $0022-23. The worse this mod can do (I hope!) is to prevent the use of byte $3FFF in the Basic stack.

I have not yet had any problems in using EZLOAD, but Murphy syas that someone out there will, and probably the first time out. I would be interested in any comments or suggestions.

```
2CB2            EZLOAD ORG   $2CB2

                ZERO PAGE LOCATIONS

2CB2            TOPL   *     $0022  TOP LIMIT OF
2CB2            TOPH   *     $0023  USER SPACE
2CB2            SPL    *     $0026  T-B STACK
2CB2            SPH    *     $0027  POINTER
2CB2            ALO    *     $0082  TINY'S
2CB2            AHI    *     $0083  VARIABLE "A"
2CB2            FLAG   *     $00F8  LOAD ON/OFF SW
2CB2            POINTL *     $00FA  POINTER FOR
2CB2            POINTH *     $00FB  LOAD ROUTINE

                KIM SUBROUTINES

2CB2            PACKT  *     $1A00  CONV ASCII/HEX
2CB2            OUTCH  *     $1EA0  OUTPUT CHAR
2CB2            INCPT  *     $1F63  INCR LOAD PTR

                SET T-B OUTPUT JMP VECTOR AT $2009
                TO ADDRESS $2CB2

2CB2 48         ENTRY  PHA           SAVE CHAR
2CB3 20 A0 1E          JSR   OUTCH   THEN PRINT IT
2CB6 C8                INY           ZERO Y-REG
2CB7 68                PLA
2CB8 C9 0D             CMPIM $0D     WAS IT CR?
2CBA F0 0A             BEQ   SETFLG  EXIT LOAD MODE
2CBC 24 F8             BITZ  FLAG    LOAD MODE ON?
2CBE 70 09             BVS   ALOAD   YES - 1ST CHAR
2CC0 30 0C             BMI   BLOAD   YES - 2ND CHAR
2CC2 C9 5C             CMPIM '\      PREFIX CHAR?
2CC4 D0 02             BNE   OUT     NO - SKIP
2CC6 85 F8      SETFLG STAZ  FLAG
2CC8 60         OUT    RTS

2CC9 06 F8      ALOAD  ASL   FLAG    TOGGLE BIT
2CCB 4C 00 1A          JMP   PACKT   1ST NYBBLE
2CCE 46 F8      BLOAD  LSR   FLAG
2CD0 20 00 1A          JSR   PACKT   CODE BYTE IN ACC
2CD3 91 22             STAIY TOPL    PARK IT
2CD5 A6 26             LDXZ  SPL     NOW DEC
2CD7 D0 02             BNE   SKIP    STACK PTR
2CD9 C6 27             DECZ  SPH
2CDB A5 27      SKIP   LDAZ  SPH     COPY TO
2CDD 85 FB             STAZ  POINTH  LOAD PTR
2CDF 85 83             STAZ  AHI     & VAR "A"
2CE1 CA                DEX
2CE2 86 26             STXZ  SPL
2CE4 86 FA             STXZ  POINTL
2CE6 86 82             STXZ  ALO
```

```
2CE8 C8        SHUFL  INY           MOVE ALL
2CE9 B1 FA            LDAIY POINTL  BYTES DOWN
2CEB 88               DEY           ONE PLACE
2CEC 91 FA            STAIY POINTL
2CEE 20 63 1F         JSR   INCPT
2CF1 A5 FA            LDAZ  POINTL  CK IF
2CF3 C5 22            CMPZ  TOPL    ALL DONE?
2CF5 A5 FB            LDAZ  POINTH
2CF7 E5 23            SBCZ  TOPH
2CF9 90 ED            BCC   SHUFL   MORE
2CFB 60               RTS           NEXT CHAR..
```

```
                      SAMPLE ORG   $0200

                      THIS IS A SAMPLE MACHINE-CODE ROUTINE
                      TO ILLUSTRATE USES OF EZLOAD

                      SET UP A NUMERICAL ARRAY OF 128
                      16-BIT ELEMENTS IN MEMORY SPACE
                      2A00-2AFF, INDEXED BY 0 TO 127


                      READ ROUTINE, R=USR(A,I), WHERE R=CONTENTS
                      OF ARRAY(I), A=ADDRESS, I=SUBSCRIPT

0200 98         READ  TYA           TRANSFER INDEX
0201 0A               ASLA          MULTIPLY BY 2
0202 AA               TAX           USE FOR INDEXING
0203 BD 00 2A         LDAAX $2A00   INTO ARRAY
0206 E8               INX           NOW GET
0207 BC 00 2A         LDYAX $2A00   HIGH BYTE
020A 60               RTS

                      WRITE ROUTINE, Z=USR(B,W,I), WHERE Z=DUMMY
                      B=ADDRESS, W=VAL TO BE STORED, I=SUBSCRIPT

020B 86 F9      WRITE STX2  $F9     PARK X FOR NOW
020D 0A               ASLA          SUBSCRIPT * 2
020E AA               TAX           USE FOR INDEXING
020F 98               TYA
0210 9D 00 2A         STAAX $2A00   STORE LO BYTE
0213 A5 F9            LDAZ  $F9     GET HI BYTE
0215 E8               INX           ..AND
0216 9D 00 2A         STAAX $2A00   STORE IT
0219 60               RTS
```

```
1 REM \980AAABD002AE8BC002A60
2 REM \86F90AAA989D002AA5F9E89D002A60
3 REM
4 REM PROGRAM TO DEMO USE OF EZLOAD
5 REM
6 REM MACHINE CODE CREATES ARRAY READ AND WRITE FUNCTIONS
7 REM BASIC PROGRAM LOADS 64 RANDOM NUMBERS AND PRINTS THEM
8 REM THEN SORTS THE ARRAY AND PRINTS THE RESULTS
9 REM
10 B=A+11:C=0
20 Z=USR(B,RND(1000),C):C=C+1:IF C<64 GOTO 20
30 GOSUB 100
40 REM SORT THEN PRINT
50 R=63
60 F=0:C=0:L=R
70 IF USR(A,C)<=USR(A,C+1)GOTO 90
80 T=USR(A,C):Z=USR(B,USR(A,C+1),C):Z=USR(B,T,C+1)
85 F=1:R=C
90 C=C+1:IF C<L GOTO 70:IF F=0 GOSUB 100:GOTO 60
95 END
100 C=0:PR
110 PR USR(A,C),:C=C+1:IF C-C/8*8=0 PR:IF C<64 GOTO 110
120 PR:RETURN
```

```
:RUN

985     633     946     338     310     186     51      816
230     248     700     186     143     65      47      456
126     831     161     173     233     681     268     869
344     477     673     609     187     981     597     496
244     58      256     541     142     917     365     183
210     263     510     333     967     420     560     145
370     774     487     919     46      838     342     614
340     606     534     318     995     326     614     695


46      51      58      126     142     143     145     161
173     183     186     186     187     210     230     230
244     248     256     263     268     310     318     326
333     338     340     342     344     365     370     420
456     477     487     496     498     510     534     541
560     597     606     609     614     614     633     666
673     681     695     700     774     816     831     838
869     917     919     946     967     981     985     995
```

assembler

ASSEMBLER FORMAT CONVERSION

Eric

Transferring Micro-Ade assembler source files over to the MOS/HDE assembler format is not very difficult. First, use the ID=FF KIM cassette read option and load the files into your text buffer (wherever that may be). Examine KIM address 17ED, 17EE and find out the location of the last byte that was loaded. Go to this address and enter a $0D, then insert a $1F (end of file marker) in the next location. Re-enter the text editor and let it know that there is an active file in the text buffer. With the HDE Editor all you do is execute a FIL A xxxx where xxxx is the start address of the active file. Both source file formats use packed BCD line numbers so at this point you can actually list the file. Oh, one more thing- the first character in the Micro-Ade file is a $0D- this must be changed to $00 also change the third character to a $20. NOW you can list the file.

From here on in it's just a matter of editing. Most of the stuff, such as CMPIM or ORG $0200 can be changed to CMP # and *=$0200 by use of the 'string search and replace' command in the HDE text editor.

Other things, such as indexed instructions and byte tables will have to be changed using the EDT (line edit) command. Don't forget to install a .END directive at the end of the file so the assembler knows when to quit.

MORE ON THE 2 PASS PATCH FOR THE ARESCO ASSEMBLER

by John Eaton
1126 N 2nd
Vincennes IN 47591

This should help to clarify the use of my two-pass patch with the Aresco assembler. The code that is needed for the $E000 version is:

```
E57A 4C F0 F0
F0F0 B1 52 A0 03 29 1F C9 10 D0 01 88 A9 01 4C 7D E5
```

In order to understand how this patch works, you must realize why we need tow pass assemblers. When you assemble a program with the original assembler you will set a listing that will generally have a lot of **'s in the machine code columns. This is because a forward reference was made to a label not in the symbol table. The assembler did not know what to do so it places a ** in the listing. Later when the label is defined it will update the object code in the machine but it cannot do anything about the lisitng. When the assembler is finished you will have an incomplete listing but the symbol table in the machine will be complete.

The assembler allows a source to be assembled in segments by assembling the first segment from $E000 and all the rest from address $E011. You can use this as a two pass assembler by assembling a source program twice. The first time start the assembler at $E000 which the "A" command will do from the editor. Then reassemble the same program a second time starting at address $E011. The first assembly will produce a complete symbol table that the second one will use. The machine code will be reproduced and copied over the first version but the important thing is that with a complete symbol table that assembler will not have to do any forward references the second time. This means no **'s.

You may wonder what happens on the second pass when the assembler encounters the labels that are previously defined in the symbol table. Fortunately the assembler is written so that you may define a label as many times as you like as long as you always define it to be the same value.

Now this sounds like a tricky way to get a clean listing, so why is a patch needed. Well the problem is caused by the way the assembler handles forward references. When you use a forward reference it must allocate enough memory space to hold that instruction. Since instructions that use memory can be either 2 or 3 bytes it always allocates 3 bytes for a forward reference. If when the symbol is defined it finds that only 2 are needed then it will fill in with a NOP.

So, if you use a forward reference for a 2 byte instruction, it will allocate 3 bytes for it. Now when the assembler is run the second time it will not see any forward references so that the instruction will be allocated 2 bytes. Every lable after that instruction will be assembled as one less than is listed in the first run symbol table and will be counted as an error.

This can only occur when you make a forward reference that assembles into a 2 byte instruction. The only instruction that do this are page zero instructions and branch instructions. You can allocate all of the page zero memory at the start of your program and no forward references will be required however the branch problem requires the patch. The patch will perform a test on the opcode that is used in a forward referenced instruction. If it is a branch then the length is forced to two bytes. Using the patch may cause some strange errors in the first pass but they all seem to come clean in the second pass. Leave the END statement out of your program until the last pass of the last segment so that the symbol table will not be printed.

WHILE LAYING OUT THIS ISSUE, I SCREWED UP AND HAD THIS PAGE BLANK. SO HERE'S A SECRET SECTION OF SOME COMMENTS WHICH I HAD PLANNED TO PUT IN THE LETTERS SECTION BUT SOMEHOW "RAN OUT OF ROOM". IT DOES MAKE ME FEEL A BIT BETTER TO KNOW THAT MY ORGANIZATION RATHER THAN MY CALCULATIONS WERE SLIGHTLY OFF.

I WANT TO WISH EACH AND EVERY ONE OF YOU A VERY HAPPY HOLIDAYS.....'ERIC

I am willing to be a "GOOD GUY" and help other members through the mail via S.A.S.E.

Bruce Davidson
Box 1738
Bismarck ND 58501

Thomas J Coyle III
11601 Dunstan Wy #301
Los Angeles Ca 90049

Dear Eric,

After reading your latest issue (no. 13) it seems that you intend to make the MOS Technology 44 pin bus the only KIM bus. That is really great if you have a MOS Technology, HDE, or Atwood 44 pin motherboard. However, the MOS Technology "K" series cards will not fit the HDE or Atwood motherboard and special mechanical adapters must be used to allow the HDE boards to plug into the MOS Technology mother board! Some standardization! The point is, if no one can agree on one specific standard, why not develop several that can be followed depending on which is best for the individual at any given time.

I propose, therefore, that there be at least two standards: (1) the MOS Technology 44 pin bus and (2) the Forethought Products KIMSI S100 bus. Both work equally well, but are obviously not interchangeable. This will allow those of us who have 44 pin mother boards to standardize our designs and software and those of us who have KIMSI S100 bus systems to standardize our software and determine which S100 boards will or will not work on the KIMSI system..

At the present I am running a KIMSI S100 system with 32K of RAM, 16K EPROM, a real time clock, and a CGRS disk controller and DOS. The CGRS disk system and SA-400 mimi-disk drive cost me only $600 which is $100 less than the HDE mini-disk system. The DOS works fine and I have had no trouble with either the controller or the drive.

I have patched the DOS into Micro Z's version of the Microsoft, KIM 9 digit Basic. The link subprogram can reside either outside of or inside of the basic interpreter. When located inside the basid interpreter, it takes the place of the Hypertape program.

The Micro Z Basic is very good and does not require the "Y" or "N" answer to the SIN, COS, TAN mode question. It is slightly larger than the Johnson Computer basic, but this is no problem. If you plan to program in Basic you should have at least 16K or more of RAM.

The new 6502 User Notes looks very good and will continue to recieve my support.

Eric

I happened to be going over some back issues of the Notes and noticed several repetitions of a misconception about video displays. Occasionally, one will hear that such a product displays 64 characters per line "or less for use with modulators." I'm presently running 64X16 characters via a VHF modulator into two different color TV's with no trouble!

The trouble is a confusion between bandwidth, resolution, and rise times in a video display system. Indeed, if you work out the math for a dot-matrix character generator you find that the highest frequency components of the video signal are just within reach of a good monochrome monitor and way beyond the normal frequency response of a modulator/TV combination.

We aren't dealing with a smoothly modulated signal, however. The video signal is a fast-rising pulse train, producing overshoot and ringing in the receiver. Although usually considered a problem, these characteristic "overdriven amplifier" conditions serve to enhance the viaual display of a video character much as the "crispening" knob on a Sony Trinitron serves to increase the apparent sharpness of a TV picture.

So, in practice, the only trouble with a 64 character line is that narrow vertical lines tend to be a bit dimmer than horizontal strokes. Careful adjustment of the receiver's fine tuning, contrast, and sharpness (if any) controls will minimize this problem.

I am presently using a XITEX SCT-100 video board and a homebrew modulator using a National LM1889 chip. I've seen other combinations that work as well.

I just ordered a copy of the FORTH Interest Group's implementation of FORTH for the 6502. It will supposedly be ready in August and I'll let you know how it works at that time.

I also received one of the Computerist's first motherboards (the Mother-Plus.) It seems to be pretty good; there's a few traces on the PC board that run mighty close to mounting nuts, etc., but it does work. One interesting thing...I bought this board as it's the only one to my knowledge that easily accepts the double edge connector format of the KIM and Bob Tripps other boards. What it does not take is an early serial number Memory plus board! Apparently, the layout designer for the first Memory Plus boards got the inter-connector spacing wrong so you have to do a bit of filing and connector moving to get the board to enter the motherboards' connectors.

What interests me about this Motherboard is that, even though it supposedly only takes 5 boards, in an actual system it may take more. If you have a messy collection of boards from various vendors using the S-44 bus, your memory, I/O, and other boards will tie up slots on both busses (for boards from the Computerist) or only on the "Expansion Bus" side (for HDE, etc.). So, this gives you several uncommitted and unwired 44 pin edge connectors on the "Applications" side that you can use to build up those utility circuits that don't connect to the S-44 bus; AC line drivers, relays for cassette control, I/O port controlled PROM burners, etc. Vector boards are available to fit with edge pins and all.

I'm presently rewiring my motherboard to take advantage of this and get out of the present "rat's nest in a box" effect.

Best Regards,

Milan Merhar
697 Boylston St.
Brookline MA 02146

interface

Gino F. Silvestri
Loral Electronic Systems
Engineering Division
999 Central Park Avenue
Yonkers, NY 10704

BROADEN YOUR I/O CHEAPLY WITH A NON-6500 PIA
or
WHO SAID YOU COULD ONLY USE 8080 PERIPHERALS WITH 8080 PROCESSORS??

With just one NAND gate and a resistor, you can use an 8255A Programmable Peripheral Interface chip to add 24 extra lines of bidirectional, handshakeable I/O to your 6500-series processor, for about $10.00!

All one really has to do is make the 6502 read/writes match those expected by the 8255A.

In this demonstration application, the 8255A is hooked up to a KIM-1, in the simplest possible manner. This simplicity results in a waste of memory space in K3 of KIM's memory map. Should you wish to preserve space above 0C03, you'll have to decode A2 through A15 to disable the PIA when using memory and vice-versa.

It is expected that this setup should work with SYM and AIM, but since these already sport nifty 6522 VIA's, only KIM's memory areas will be mentioned.

Very briefly describing the use of the 8255A, (Radio Shack supplies a 12-page "manual" with the chip) we see that there are chip select, read/write and reset lines similar to those used in devices such as the 6520 PIA. Also in looking at the 8255A diagram, one sees similar bidirectional DATA lines to the 6500 series. But it's at the I/O pins that this 40-pin monster shows its stuff! The 8255A has 24 (count 'em-24) available I/O pins. Their functions may be chosen by an amazingly complicated set of instruction formats sent to the mode select or control register at 0C03.

Depending on variations in the format of this control word, the 24 pins are split up into 3 or more groups. Most commonly used are the groups in which the A,B, and C ports are split into units of 8 lines each, arranged as 8 in with 16 out, 24 in, 24 out, or similar combinations. In addition, port C may be split in half giving a 4/4 line fraction to these 8 line groups. Note that the 8255A is not programmable for individual line input/output as are the 65xx series devices.

Variations of the control modes yield strobed or edge triggered handshake/acknowledge lines; various combinations of simultaneous bidirectional ports; and a unique mode allowing the setting or resetting of 8 individual lines on port C by decoding 3 bits of the control register-as in a one-of-eight decoder/selector!

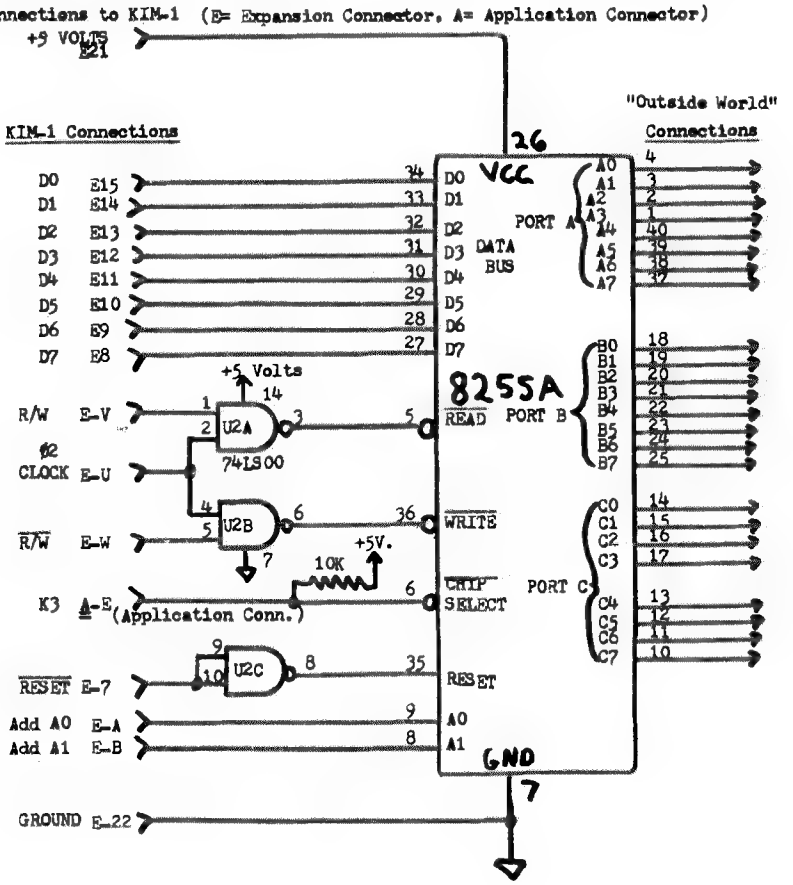
The 8255's reset line behaves in a manner similar to 65xx devices-bringing all outputs to a tri-state condition. This also resets the mode register-so be sure any application you have restores the control word after a reset.

Space cannot allow further description of this versatile device-the National Semiconductor manual provided by Radio Shack with the chip, or an Intel catalog will be required to provide full details. However, here's a brief application program for the KIM-1 to demonstrate one of the 8255A modes:

---ALL PORTS BECOME OUTPUTS FOR DATA---

```
0200 A9 80      LDA#  Code for all ports="OUTPUT"
0202 8D 03 0C   STAabs 0C03=Control register
0205 A9 xx      LDA#  user data for Port A out.
0207 8D 00 0C   STAabs 0C00=Port A
020A A9 xx      LDA#  user data for Port B out.
020C 8D 01 0C   STAabs 0C01=Port B
020F A9 xx      LDA# user data for Port C out.
0211 8D 02 0C   STAabs 0C02=Port C
0214 00         BRK
```

Connections to KIM-1 (E= Expansion Connector, A= Application Connector)



KIM AUDIBLE WARNING INTERFACE

by H. T. Gordon
641 Paloma Ave.
Oakland, CA 94610

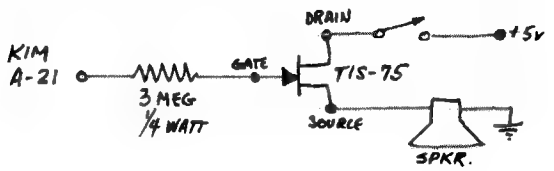
This uses (with a tiny bit of hardwaring)toggling of the 6530-002 PA0 for tone production, leaving the 6530-003 output port free for more important work. It uses 4 locations in the KIM-reserved area of zero-page, that are not normally in use when KIM is "singing". Both the duration and the frequency of the tone are controlled by a single byte, program-set in 00F7 before the JSR SINGER, and not altered by the operation. Locations 00F6, 00FD, and 00FE are used as working registers, but need no setting and are all zeroed when the signal ends. They control 3 loops, and a call to SINGER does not alter the X- or Y-registers. The coding (with instructions numbered in parentheses) is:

SUBROUTINE "SINGER"

```
(1)   A9 01      (sets 6530-002 PA0 as
(2)   8D 41 17    an output)
(3)   A5 F7      (LDA the pre-set control number)
(4)   85 F6      (STA into 00F6)
(5)   85 FD      (STA into 00FD)
(6)   49 FF      (EOR#FF complements accumulator)
(7)   85 FE      (STA into 00FE)
(8)   EE 40 17   (toggles PA0 by an INC)
(9)   C6 FE      (decrement 00FE, the frequency con-
                   trol)
(10)  EA EA EA   (sequence of 3 NOPS, extra frequen-
                   cy control)
(11)  D0 F9      (if 00FE not zero, back to (9))
(12)  C6 FD      (decrement duration control number
                   in 00FD)
(13)  D0 F0      (if not zero, back to (7))
(14)  49 FF      (regenerate control number in ac-
                   cumulator)
(15)  C6 F6      (decrement duration control number
                   in 00F6)
(16)  D0 E6      (if not zero, back to (5))
(17)  60         (RTS)
```

The 3 NOPS at (10) are not strictly necessary, but (if the subroutine is in RAM) can be overwritten by one of 3 "neutral" JSRs to the KIM-ROM that have no effect on the processor status but prolong the fundamental timing of the innermost loop. Durations are prolonged about 1.4X by 20 48 19, about 1.7X by 20 48 19, and about 1.9X by 20 45 19. Whichever of the 4 options is used, tone frequency is lowest and duration longest with an 00F7 value of FF or 00. Frequency rises from 01 to FE. Duration is short at either end, increasing to a near-plateau (about 35 seconds for the 3-NOP option) in themidrange from 90 to C0. One can control duration best at the low and high range of frequencies, or obtain relatively constant duration and vary the frequency in the midrange. The upper audible limit is about F7 for the 3-NOP option, somewhat higher for the other options; higher frequencies are more attention-getting and so better for warnings. For use as a simple time delay (without sound) enter SINGER at instruction (3).

HARDWARE ADDITION TO KIM-1. The following circuit provides KIM-1 with its own voice, a miniature 2.75" PM 8-ohm speaker. Users who have modified their audio cassettes for use as an audio amplifier can get much louder sound by connecting the JFET source to its audio input. The JFET is an inexpensive surplus TIS-75. The switch is optional; KIM LED displays in which PA0 in active cause a hum, that can be switched off if the user finds it annoying.



KIM 4K EXPANSION NOTE

by Dr. R. J. Allen
Groningnen
Netherlands

I would like to point out that, contrary to the "Note" on page ∎ of the KIM-2 Users Manual, this 4K board can easily be inserted into the memory block 0400-13FF with only two extra resistors, as follows:

-do not connect pin 16 of KIM-2 to KIM-1 pin AK; leave AK jumpered to ground, and pin 16 simply disconnected.

-do not connect pins S, T, U on KIM-2 to KIM-1 at all; instead, tie them together at the connector, then through a 1K pull-up to +5V.

-Wire-OR the KIM-1, K1, K2, K3, and K4 decoder outputs (U4) together (AC, AD, AE, AF on KIM-1), and then through a 2K pull-up to +5V. Connect the common tie point of the four decoder outputs to pin R on KIM-2.

-Set the on-board DIP switches S1, S2, S3, S4 to off, off, off, on (note that the description on Fig. 3 of the KIM-2 Users Manual as to the appearance of the DIP switch seems to be just opposite to what it should be).

INTERFACING THE TVT-2 VIDEO BOARD WITH THE KIM-1

by W. C. Clements, Jr.
Chemical Engineering
Univ of Alabama
Box 2662
University, Al 35486

Those of us who are not fortunate enough to own a hard-copy TTY often choose one of Don Lancaster's video display units as an alternative. His TVT-6 is very popular and in wide use these days, but the older TVT-2 with the serial interface adapter (SIA) option, (although larger and more expensive) does it all with hardware, tying up neither KIM memory nor input-output ports. The display is a clean, snow-free 16 line by 32 character display. The only trouble is - its serial interface produces RS-232 signals wigh a wide variety of baud rates and parity/bit number choices, while the TTY input on KIM wants 20 ma. current-loop signals with Teletype Corporation ASR-33 compatibility. Also, the older keyboard (the KBD-2) which Southwest Technical Products Corp. used to furnish with its TVT-2 kit, has no RUBOUT key. Overcoming these differences took a bit of experimenting, but the results are well worth the trouble.

The first order of business is to arrange for RS-232-to-20 ma.-interfacing. Although a number of simple interface circuits have been published, I chose a slightly modified version of the circuit given in 6502 User Notes No. 4 and also in Pyramid Data System's "XIM User Manual."[1,2] The original circuit would not drive the TVT-2's RS-232 input, but a simple resistance change fixed the problem (see Figure 1). The transistors can be any general purpose silicon types that will handle 12 volts. I used a 2N2222 for the NPN and a 2N5139 for the PNP. This circuit places KIM's TTY KBD input at +5v. for a RS-232 signal of -12v. (logical one) and at ground for a RS-232 signal of +12v. (logical zero). The interface was built on a small piece of perf-board and mounted with a fiber standoff at the upper right-hand corner of the TVT-2's SIA board-there is room for one small hole, carefully drilled, just to the left of diode D7. That board is crowded! +5v. and -12v. are taken from the same board, as indicated on Figure 1.

It was not clear, from reading the KIM manuals, what form the bit stream into TTY KBD should take. The TVT-2 serial interface provides a number of combinations for parity type and bit number, depending on installation of jumpers D through K on

the SIA board. The KIM TTY monitor was found to operate properly with no jumpers installed, providing no parity, 8-bit code, bit 8 = 1. (I also use my TVT-2 with a Pennywhistle 103 modem to access The University of Alabama's Univac 1110 system through its dial-up ports, so I used a switch to provide even parity with no bit 8, as an option.)

If the KBD-2 keyboard is used, it must be provided with a RUBOUT key. This is easily done by using one of the uncommitted keys, as shown on Figure 2. For those with other keyboards, a study of its circuit diagram should show how to provide RUBOUT (ASCII $FF) if it is not so equipped already.

The system described above allows me to handle I/O from the KIM built-in monitor with no loss of memory space or use of the application ports. It also works beautifully with Pyramid Data System's XIM program, which provides an extended set of TTY commands for users with 1K of additional memory. If you want graphics, or need a denser screen of text, MTU's Visible Memory board will give you a video screen of 64,000 dots to work with, in an 8K expansion board. Two of these, plus the TVT-2, provide 16K of expansion memory and three independent video displays in my system.

Incidentally, SWTP's SIA board provides all standard baud rates between 110 and 1200 baud for those willing to add a crystal and a few other parts. My KIM works fine at all these data rates, in contrast to reports in the literature of troubles at rates over 300 baud. A hex dump at 1200 baud does require a quick trigger finger on the reset key!

References

1. Kim-1/6502 User notes, No. 4, p. 3.

2. "XIM Extended I/O Monitor for the KIM-1, "Pyramid Data Systems, New Egypt, N.J., p. 6.

KIM BATTERY BACKUP

Lauren Kline
3596 Beacon Dr.
Beachwood Oh 44122

I have installed a backup power source which is automatically switched in and now a momentary power interruption won't scramble KIM-1's brain. I used D-cell sized 4 amp hour NICADS. As the fully charged terminal voltage is 1.45 volts three (3) cells yield 4.5 volts approximately. This seems to be enough to keep things cooking. See the attached schematic for the hook up details.

NOTE 1 - Regulator must be reset to give 5 volts DC at the KIM-1 power input terminal. There is some voltage drop across a diode.

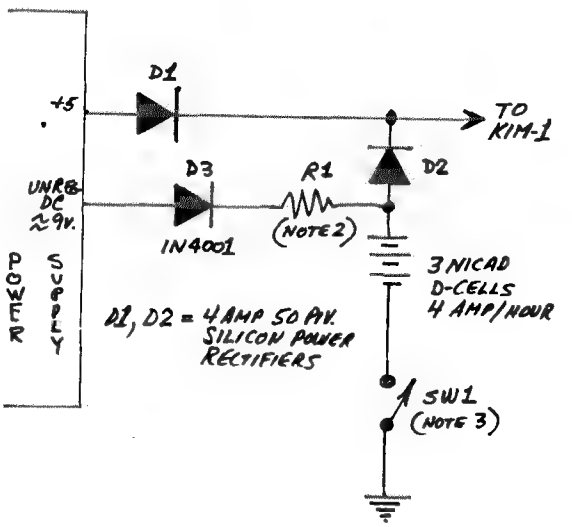
NOTE 2 - The value of resistor R. determines the charging rate to the NICADS. This will vary with the size and type of NICADS used.

$$R. = \frac{\text{UNREGDC - BATT VOLTS}}{\text{CHARGING RATE I}}$$

NOTE 3 - The switch allows disabling of the battery. It could be an additional pole on the AC ON/OFF switch if desired.

The diode (D_1) in the 5 volt bus from the power supply prevents the regulator form loading the battery during power loss the diode D_3 in the charging circuit serves the same purpose. The diode D_2 disconnects the battery from the bus under normal conditions.

The same idea could be implemented using high current alkaline D-cells if desired. Just delete the charging circuit, as alkaline cells cannot usually be recharged.



CASSETTE stuff

```
;TAPE LOAD DISPLAY ON KIM LEDS

;FROM FRANK HOGG
;       204 WINDEMERE RD.
;       SYRACUSE, NY 13205
;       315-469-4811

;LOAD MEMORY FROM TAPE WITH DISPLAY
;ON LEDS LIKE MICRO-ADE ASSEMBLER.

;THE LEDS WILL DISPLAY THE FOLLOWING:

;WHILE THE KIM IS LOOKING FOR DATA,
;A FLICKERING 8 IS DISPLAYED IN THE
;RIGHTMOST LED.

;THE SYNCH CHAR IS DISPLAYED AS THE
;RIGHT TWO VERTICALS AND LEFT LOWER
;VERTICAL

;THE DATA IS DISPLAYED AS THE TOP
;TWO VERTICALS AND THE BOTTOM HORI-
;ZONTAL.

;IF THE SYNCH COMES ON THEN GOES
;BACK TO THE THE 8, THEN KIM DID
;NOT READ THE PROPER ID NUMBER
;AND WILL CONTINUE SEARCHING.
;IF YOU HIT RESET, THE ID THAT
;WAS READ FROM THE TAPE CAN BE
;DISPLAYED BY EXAMINING LOCATION
;$0000.

;PUT YOUR SEACH ID AT LOCATION
;$1780, HIT '+' AND GO AT $1781.
;LOCATION $00F1 IS SET TO $00 BY
;THIS PROGRAM.

DECMDE =$00F1
SBD    =$1742
SAVX   =$17E9
VEB    =$17EC
PAD    =$1740
PADD   =$1741
INTVEB =$1932
RDBYT  =$19F3
RDCHT  =$1A24
RDBIT  =$1A41
```

```
01-0240  2000                    *=$1780
01-0241  1780                    .OFF 2780
01-0245  1780                ID  *=*+1            #SEARCH ID GOES HERE
01-0250  1781
01-0255  1781  A9 00      START  LDA #0           #SET UP FOR DECIMAL
01-0260  1783  85 F1             STA DECMDE       #MODE
01-0265  1785  A9 7F             LDA #$7F         #TURN ON LED
01-0270  1787  8D 41 17          STA PADD         #BY SET DD REG
01-0275  178A  D8                CLD
01-0280  178B
01-0285  178B                #THIS IS LIKE $1873 ON KIM
01-0290  178B
01-0295  178B  A9 8D      XLOADT LDA #$8D         #INIT VOLATILE EXEC
01-0300  178D  8D EC 17          STA VEB          #WITH STA ABS.
01-0305  1790  20 32 19          JSR INTVEB
01-0310  1793
01-0315  1793  A9 13             LDA #$13         #TURN ON CASSETTE HARDWARE
01-0320  1795  8D 42 17          STA SBD
01-0325  1798
01-0330  1798  A9 4C             LDA #$4C         #JUMP TYPE RETURN
01-0335  179A  8D EF 17          STA VEB+3
01-0340  179D  A9 0F             LDA #$0F
01-0345  179F  8D F0 17          STA VEB+4
01-0350  17A2  A9 19             LDA #$19
01-0355  17A4  8D F1 17          STA VEB+5
01-0360  17A7  A9 FF             LDA #$FF         #CLEAR SAVX FOR SYNC AREA
01-0365  17A9  8D E9 17          STA SAVX
01-0370  17AC
01-0375  17AC  20 41 1A  XSYNCA  JSR RDBIT        #GET A BIT
01-0380  17AF  4E E9 17          LSR SAVX         #SHIFT BIT INTO CHAR
01-0385  17B2  0D E9 17          ORA SAVX
01-0390  17B5  8D E9 17          STA SAVX
01-0395  17B8  8D 40 17          STA PAD          #PUT IT ON LED
01-0400  17BB  C9 16      TST    CMP #$16         #IS IT A SYNC CHAR?
01-0405  17BD  D0 ED             BNE XSYNCA       #NO, TRY AGAIN
01-0410  17BF  20 24 1A  XSYNCB  JSR RDCHT        #IN SYNC ?, READ A CHAR
01-0415  17C2  8D 40 17          STA PAD          #DISPLAY ON LED
01-0420  17C5  C9 2A             CMP #$2A
01-0425  17C7  D0 F2             BNE TST          #IF NOT, LOOP AGAIN
01-0430  17C9
01-0435  17C9  20 F3 19  XLOADK  JSR RDBYT        #READ ID FROM TAPE
01-0440  17CC  85 00             STA $0000        #STORE FOR YOUR INFO
01-0445  17CE  CD 80 17          CMP ID           #COMPARE WITH REQUESTED ID
01-0450  17D1  F0 0D             BEQ KIM          #YES, THEN LOAD IT
01-0455  17D3  AD 80 17          LDA ID           #WHAT ABOUT DEFAULT $00
01-0460  17D6  C9 00             CMP #$00         #IS IT $00?
01-0465  17D8  F0 06             BEQ KIM          #THEN LOAD IT
01-0470  17DA  C9 FF             CMP #$FF         #DEFAULT $FF? IGNORE TAPE SA
01-0475  17DC  F0 05             BEQ KIMFF        #YES, THEN LOAD TO ADDR $17F
01-0480  17DE  D0 CC             BNE XSYNCA       #NO ?, THEN TRY AGAIN
01-0485  17E0
01-0490  17E0  4C D7 18  KIM     JMP $18D7        #LOADT5 IN KIM ROM
01-0495  17E3  4C EC 18  KIMFF   JMP $18EC        #LOADT6 IN KIM ROM
01-0500  17E6
01-0505  17E6                    .END
```

CASSETTE SAVE USING ALTERNATE STARTING ADDRESS

by Philip K. Hooper
3 Washington St.
Northfield VT 05663

Occasionaly it can be useful to read a cassette file into a memory block other than the one from which it was dumped. For the first file on a tape, this is easily accomplished using the load ID 'FF'. The procedure below permits placing onto tape, during a dump, a starting address DIFFERENT from the one at which the code being dumped actually resides, and hence permits reading that code back in at the alternate address. (This might be useful, for example, if one intended to subsequently reload the file into an unused realm of memory and later transfer selected portions of it to its normal residence; or for using a page-one staging of Hypertape to record a program that is intended to reside, later, in page one; or for other sorts of memory conflicts that are temporary consequences of some program development stage.)

Let SAL, SAH, EAL, EAH, ID have their usual I/O interpretation, and let RAL and RAH stand for the low and high bytes of the 'recall' address, the address you wish to have recorded on tape as the starting address.

Enter the following values:

```
17E7 00
       clear checksum
17E8 00

17EC  AD

17ED SAL
     actual code location
17EE SAH

17EF 60

17F5 RAL
     recall address
17F6 RAH

17F7 EAL as for

17F8 EAH an ordinary

17F9 ID dump
```

Then 'GO' from 1808 (0108 Hypertape). This bypasses the normal initialization routine which moves 17E5,6 into 17ED,E.

Although the contents of 17F5,6 will be written to the tape as the starting address, the values keyed into 17ED,E will point to the first byte of code that is fetched for dumping to tape.

AIM 65. The head start in educational microcomputers.

On-Board Printer, Advanced R6502 CPU, Versatile I/O — It's the Honors Candidate for Microprocessor Learning

Micro Power

It's tops in its class because it's expressly designed for microprocessor learning. Rockwell's AIM 65 is a fully-assembled microcomputer system with special educational features at a low price school budgets can afford.

AIM 65's on-board thermal printer — unique in its price range — produces hard copies of exercises for easy checking by student and instructor. On-board I/Os provide dual cassette, TTY and general purpose interfaces. Bus and system expansion is built in. Same for PROM, ROM and RAM expansion.

AIM 65's Interactive Monitor prompts students each step of the way in hands-on learning of microprocessor fundamentals. It includes a Text Editor, Interactive Mnemonic Assembler, Debugger (Trace, Breakpoints), and more!

An optional fully symbolic Assembler program makes AIM 65 a powerful hands-on learning system for microcomputer development and prototyping. Advanced students can explore high level languages with an optional ROM-resident BASIC Interpreter. There's even a college textbook available.

And you'll find AIM 65 is ideal for equipment control and other laboratory computer applications.

Discover how with one low investment you can combine several AIM 65s for hands-on, high-productivity microprocessor learning in classes where students don't have to wait in line. Check the high features and low prices of Rockwell's AIM 65 printing microcomputer.

Contact your local distributor or write or call AIM 65 Marketing, Electronic Devices Division, Rockwell International, P.O. Box 3669, D727, Anaheim, CA 92803, (714) 632-3824.

Rockwell International

where science gets down to business

AIM info

AIM PRINTER MODIFICATIONS

Jody Nelis K3JZD
132 Autumn Dr
Trafford PA 15085

If the columns on your printer are wavy, you may benefit from a factory recommended modification which usually cures this problem. Print 20 rows of 20 "I"'s & check for straight columns.

If yours wave noticably, add two jumpers on the back of the main board as follows:

From Z36 Pin 10 to Z20 Pin 6
From Z36 Pin 15 to Z20 Pin 3

This modification adds pull up resistors to the output pins of a flip flop circuit to improve its stability.

I've made a second change which has improved the quality of my printer. It's too soon to tell if there are any adverse side effects though, so, if you want to try it, beware!

My printer printed too light. Even with VR2 adjusted to the maximum, All I got was a pale blue on white. I couldn't get enough contrast to make it easily readable.

To cure this, I replaced the 2K pot at VR2 with a 5K pot. This allowed me to up the voltage to the thermal print heads to about 22 volts. The manual says this should be 18-20 volts but this is probably an actual peak voltage. I now measure 22 volts at Pin 6 of Connector J2 while doing a memory dump to the printer ("D" Command). This is an average voltage since the VTVM I have isn't fast enough to measure peak voltage.

Anyway, I now have a crisp, clear contrast on my thermal tapes with no apparent overheating of the print head. The long term affects are yet to be seen.

I guess I'm hard on printers. First I try to sand the heads down with an abrasive paper and then I try to melt them down with more than the recommended voltage!

AIM65 BASIC--DATA SAVE/LOAD SCHEME

Steve Bresson
1666 Independence Ct
Severn MD 21144

I liked Christopher Flynn's idea of being able to read and write arrays from Basic (issue #15), but decided it was too limited. So I attempted to extend his idea on the AIM65 Basic. The pointer locations for the AIM were different, but easily found from his description. Since the AIM uses a block structured tape format, it can easily accomodate the differing data types and extra processing time that they would incur during the save/load. But this quickly got out of hand, so I determined to crack Basic and try to use some of its search routines to save space. After disassembling all of Basic and partially decoding some of it (whew! not an easy job!), I discovered the following:

1) The LOAD command does only one thing--a jump to WHEREI, in the monitor, which does the set up for any input device. If you specify tape, all input comes from tape until a <ctl-Z> is encountered, at which point Basic forces a change back to the standard input and output.

2) The SAVE command calls WHEREO ($E871) and then LIST's the program to tape. (i.e. source f form, not the compressed form which some of the other basics use). WHEREO sets up the output device and sets the output flag to the appropriate value.

When I saw this, I decided to discard the assembly language routine and try to do the job from Basic. If it worked, it would entail no hardware, I would not have to fool around with a machine language program each time I wanted to save/load, and it could be incorporated into only those programs that really needed it, rather than being resident at all times. As a simple test, I saved a text file on tape with a <ctl-Z) as the last line of the file. The following program was then run:

```
10 LOAD
20 INPUT A$
30 PRINT A$
40 GOTO 20
```

This read in the tape and echoed it to the display. When it reached the <ctl-Z> it was forced back to the standard input, and waited for keyboard input. Success!! But be careful! INPUT still expects its input to be terminated by carriage returns, and commas between multiple arguments.

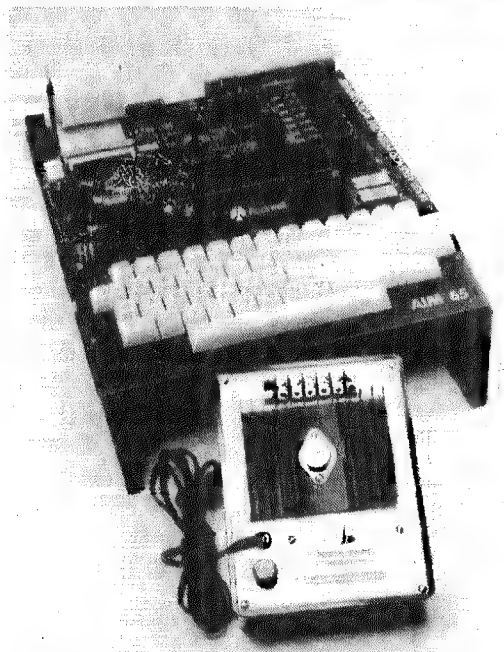
A friend and I tested a program to write to tape, from Basic, by using POKE and USR to call up WHEREO and DU11($E50A). DU11 outputs the last block to tape, shts off the oscillator (VIA), and returns you to the standard input/output. The following subroutines are a direct result of that test. The "#" is output so you can differentiate between text/basic, object, and Basic data files easily.

(<" "> =text/basic,<CR> =object, <"#",CR> = basic data)

```
2000 REM SET UP FOR BASIC DATA LOAD  8/6/79 slb
2005 LOAD
2010 INPUT ZZ$: IF ZZ$+"#" THEN RETURN
2015 PRINT!"**NOT A BASIC DATA FILE**"
2020 PRINT! ZZ$: GOSUB 2080 :REM RESET TO STANDARD
     I/O
2025 STOP
2030 RETURN

2050 REM SET UP FOR BASIC DATA SAVE 8/7/79 slb &
     wjs
2055 POKE 41993,48: REM SET UP INTER-BLOCK GAP
2060 POKE 4,113: POKE 5,232: REM WHEREO($E871)
2065 X=USR(1): PRINT "#": RETURN

2070 REM CLOSE BASIC DATA FILE
2075 PRINTCHR$(26);CHR$(13);CHR$(13)
2080 POKE 4,10: POKE 5,229: X=USR(1): REM DU11
     (E50A)
2085 RETURN

99   REM EXAMPLE SAVE USING BASIC SUBR.
100  GOSUB 2050 : REM OPEN OUTPUT FILE
115  REM OUTPUT NOW GOES TO TAPE/PRINTER/PAPER
     TAPE/...
120  FOR I=1 TO 5
125  PRINT SQR(I): PRINT "OK";I
130  NEXT I
140  GOSUB 2070 : REM CLOSE OUTPUT FILE
150  PRINT! "DONE!"
160- END
170  REM BY STEVE BRESSON & BILL SEMANCIK

200  REM EXAMPLE LOAD USING BASIC SUBR
210  GOSUB 2000 : REM OPEN INPUT AND CHECK FILE
     TYPE
220  FOR I=1 TO 5
230  INPUT J: INPUT J$
240  NEXT J
250  PRINT!"DONE!"
260  END
270  REM WHEN THE CTL-Z IS ENCOUNTERED, INPUT WILL
280  REM REVERT BACK TO THE KEYBOARD.
```

With this you now have the capability of saving and loading strings and data (in text form) form Basic.

POWER YOUR AIM-65* SAFELY WITH THE MTU K-1000-5



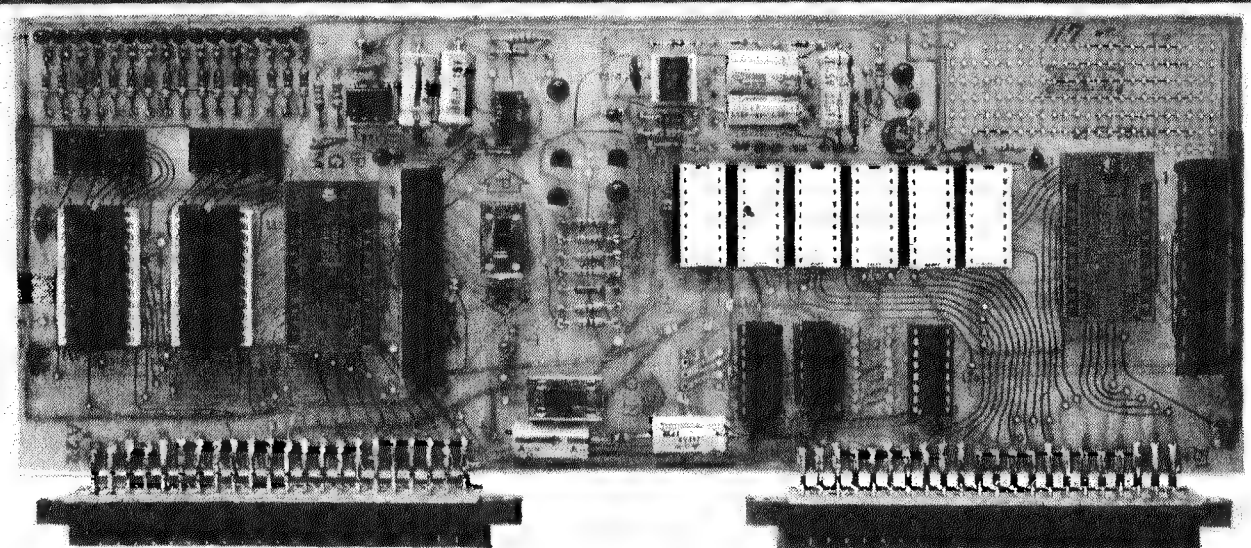
- PREFERRED MOST BY EDUCATIONAL, HOME, AND TABLE TOP USERS
- ATTRACTIVE, FULLY ENCLOSED, NO AC EXPOSED
- AC LINE CORD AND PRIMARY FUSE PROVIDED
- BARRIER TERMINAL STRIP FOR DC POWER CONNECTIONS
- 5V & 24V REGULATED FOR AN EXPANDED AIM-65
- 8V & 16V UNREGULATED FOR ADDITIONAL SYSTEM EXPANSION
- POWERS ALL MTU AIM-65 EXPANSION PRODUCTS
- $80.00, QUANTITY AND DEALER DISCOUNTS AVAILABLE

CALL OR WRITE FOR OUR FULL LINE CATALOG OF PRODUCTS

MICRO TECHNOLOGY UNLIMITED

841 GALAXY WAY
PO BOX 4596
MANCHESTER, NH 03108
(603)-627-1464

* AIM-65 IS A TRADEMARK OF ROCKWELL INTERNATIONAL



The new MORE™ board by T.T.I. is an easy to install and use expansion for the basic KIM,* AIM,** or Kim compatible micro-computer. One unique feature of this board is it's ability to program, run or copy industry standard EPROM's--2708, 2716 (+5 & +12) or 2716, 2758, TMS 2516 (+5V only). Individual program and run personality keys and software allow the user to program from RAM or copy data from any given EPROM into any other type EPROM. (Example: Empty two 2708's into one 2716!). Additionally, the board has sockets for 3K of RAM (2114's), and two zero insertion force EPROM sockets. Also featured is a 16 Bit latchable buffered output port with two dip headers for access. Associated with this port is a row of 16 LED's arranged in binary sequence. All voltages necessary to run and program the EPROMS are generated on board. Only +5 and +12 volt supplies are required. (Approximately 200 MA from each supply).

Standard 22 pin edge connectors allow the board to be plugged onto the KIM* or Kim compatible machine. All signal lines are passed through the MORE™board and are made available in total on standard 22 pin paddle cards.

The MORE™board and KIM* or equivalent, make a low cost and excellent dedicated controller, educational tool, hobby computer expansion or development system.

The price with prepaid shipping in U.S. is $169.95-includes 8 personality keys, documentation, and software listings. Options---software on tape for AIM** or KIM* $10.00---Software in 2708 EPROM for $30.00.

*Product of MOS Technology **Product of Rockwell

T.T.I.
P.O. Box 2328
Cookeville, TN. 38501
615-526-7579

AIM CASSETTE MOD

B. Strandtoft
Mollebakken 27
DK6400 Nordborg
DENMARK

"...The Audio Cassette Interface seems to be insensitive. You may improve the sensitivity by soldering a 4700 pF condenser between pin 7 and 8 of Z8 (LM311) and eventually readjust VR1. This modification cures the high-frequency oscillation tendency of Z8 and the required signal level from the Taperecorder may drop to 30 mV. This modification has been carried out on four AIM65, all with improvements in performance.

At the present I am trying to modify MICRO-CHESS for use in AIM65 but I have some difficulties because most of page 1 is occupied. Hopefully, I will also have TINY BASIC running in a short while.."

I hope that you can use it for the Notes, and that you will find space to mention our presence.

The lack of a danish magazine makes it very difficult for us to get in touch with 6502 users, and due to a poor representation of the manufactures, we sometimes feel very lonely.

SYM

TINY BASIC FOR SYM

Gunnar List
Lisco
Aprilvaenget 6
6000 Kolding
Denmark

May we inform you of our existence.

We are a small and efficient (sic) company, with no production of our own (allmost), and totally dedicated to serving personal users of 6502 systems.

A price list is included with this letter, so that you may see, by product name, what we are able to supply.

Our favorite system is the SYM-1, to which I believe you allready have received a Hypertape load routine from a friend of mine.

Included with this letter, is a description of how we have modified Tiny BASIC to run on SYM, and included a small Dump/Load feature, that can be called by USR.

Tiny BASIC for SYM-1

Dump/Load feature

```
X8C7  20 86 8B  DUMP    JSR   ACCESS
X8CA  A2 03             LDXIM $03
X8CC  9D 4B A6  SETP2   STAX  PARM2
X8CF  B5 1E             LDAX  $001E
X8D1  CA                DEX
X8D2  D0 F8             BNE   SETP2
X8D4  E8                INX
X8D5  B5 24     SETP1   LDAX  $0024
X8D7  9D 4A A6          STAX  PARM1
X8DA  CA                DEX
X8DB  10 F8             BPL   SETP1
X8DD  A0 80             LDYIM $80
X8DF  20 87 8E          JSR   DUMPT
X8E2  90 15             BCC   EXIT
X8E4  20 86 8B  LOAD    JSR   ACCESS
X8E7  8D 4E A6          STA   ID
X8EA  A0 80             LDYIM $80
X8EC  20 78 8C          JSR   LOADT
X8EF  B0 0C             BCS   EXIT
X8F1  A5 FE             LDA   EAL
X8F3  85 24             SAT   $0024
X8F5  A5 FF             LDA   EAH
X8F7  85 25             STA   $0025
X8F9  A9 00             LDAIM $00
X8FB  A0 00             LDYIM $00
X8FD  4C 9C 8B  EXIT    JMP   NACC
```

To dump a memory image of your program, key in:

A=USR(10439,1)

and the program will be written with ID-01, after the usual 8 second delay.

Load the program again by:

A=USR(10468,1)

A will be returned with the value zero, if OK, and -1 if error.

OSI 22

Dear Eric,

Thank-you for your personal assistance while constructing the 32K RAM board (6502 User Notes #15), article by J. C. Willimas. Also my thanks for the information on the alternate source for the OSI prototyping board (6502 User Notes #15), article by R. F. Solomon.

The board appears compatible with the OSI 48 pin bus and the C-2-4P with your suggested modification!

The only modification the C-2-4P needed was and additional small 12V @ 500ma. power supply, with I mounted inside the cabinet of the machine. The modification to the RAM Board was an additional IC, a 7400, tied into the Data Direction pin (B4 on the OSI bus).

I used pots for R1, R2, and R3 to set up the timing and without the use of a scope was quite tricky.

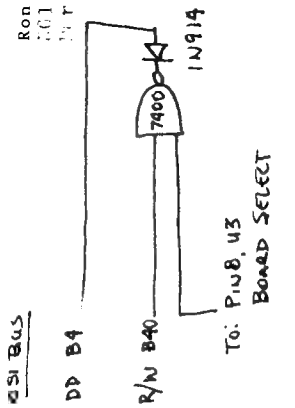
Through information obtained from your publication and from it's Readers, I am currently running 36K in my C-2-4P with a very little outlay of cash!

Hoping to see more information on the OSI machines in the future and hopefully your publication and it's readers will shed some light on the poorly documented OSI equipment.

P.S. I also take this opportunity to thank Mr. Williams and Mr. Solomon for their articles, which started the entire undertaking.

Gratefully yours,

Ron Regal
[illegible]



ZERO-PAGE MAP FOR BASIC IN THE C2-4P by Edward Carlson
3872 Raleigh Dr., Okemos, MI 48864

Enclosed are two copies of the beginning of a memory map of page $00. If you have anything to add or correct about any entries, I would very much appreciate getting your comments. If I get a substantial amount of information in this way, I will send out a second, more complete map. I also would like information on pages 01 and 02, and any useful POKEs or other program ideas.

Page $00 after a cold start. C2-4P with 16K memory and a BASIC-IN-ROM Version 1.0 Rev. 3.2.

```
00  4C 74 A2    JMP to BASIC ROM
03  4C C3 A8    JMP to BASIC ROM
06  05 AE       INVAR
08  C1 AF       OUTVAR
0A  4C 88 AE    JMP to USR(X) in BASIC ROM
0D  00 00       ?
0F  48 33       Concerns terminal line length
11  00 40       $4000 is address of first non-BASIC memory
                later, contains line number of line in buffer.
13  00 --       $13 is start of line buffer
  ...
57  --          End of line buffer
58  -- -- --    Concerns error messages?
5B  22 22       Enter (delete) line RETURN, BREAK M, then
                contains line number.
5D  -- --       ?
5F  FF --       ?
61  00 --       ?
63  -- 00       ?
65  68 65       ?
67  00 06       ?
69  92 A1       Address in BASIC ROM.
6B  -- --       ?
6D  -- --       ?
6F  -- --       ?
71  92 A1       Indirect address in BASIC ROM.
73  47 9B       ?
75  -- --       Used with RND(X)
77  -- --       ?
79  01 03       Address of start of source program in RAM
7B  03 03                         single variable table
7D  03 03                         array variable table
7F  03 03                         empty BASIC memory
81  FF 3F                         concatenated array strings
83  -- --       Address of start of concatenated single strings.
85  FF 3F                         non-BASIC memory
87  -- FF       Current line number.
89  -- --       ?
8B  -- 00       ?Address of current BASIC line?
8D  -- --       ?
8F  00 03       Current address in DATA statements.
91  -- --       An address, about DATA statements?
93  -- --       ?
95  12 --       Address of length in a string variable?
97  -- --       Address of (current?) variable.
99  -- --       ?
9B  -- --       ?
9D  -- --       ?
9F  -- 03       ?
A1  4C -- 00    JMP to an address in BASIC ROM.
A4  -- --       Addresses in page 02?
A6  -- --          "
A8  FE 00       ?
AA  -- --       ?
AC  06 92       ?
AE  68          FACHI
AF  00          FACLO
B0  20 --       ?
B2  00 80       ?
B4  00 00       ?
B6  10 00       ?
B8  92 A1       Address in BASIC ROM.
BA  98 A1       Address in BASIC ROM.
BC  E6 C3       INC LO byte of address of BASIC line; This
                is the start of a subroutine to go through
                a line character by character.
BE  D0 02       BNE

C0  E6 C4       INC HI byte if needed
C2  AD 00 03    LDA with a character of the line.
C5  C9 3A       CMP #$3A  Is it a colon?
C7  B0 0A       BCS : is a statement ender, branch to RTS
C9  C9 20       CMP #$20  Is it a space?
CB  F0 EF       BEQ  If yes, go get another character.
CD  33          SEC set carry
CE  E9 33       SBC #$33
D0  38          SEC
D1  E9 D0       SBC #$D0
D3  60          RTS  End of subroutine.
D4  80 4F       ? Varies during running of program.
D6  C7 52          "
D8  --          Stays empty
  ...
FB  00 00       Monitor
FD  --             "
FE  FE 00          "
```

PRODUCT ANNOUNCEMENT

Los Alamos NM - TIS, the company that offers a full line of workbooks and software packages for the Commodore PET/CBM computers, announces a new product for the Ohio Scientific Challenger 1P. Getting Started With Your Challenger 1P introduces the fundamentals of C1P BASIC and explains its characteristics, limitations and useful features. This document discusses calculator and program mode, input and output, data representation, and program storage on cassette.

Getting Started With Your Challenger 1P also describes C1P control and logic including testing and branching, subroutine use, and logical operations. This well-written beginner's workbook contains many exercises and sample programs throughout. It is available from your dealer or by writing to TIS, P.O. Box 921, Los Alamos, NM 87544. Price is $5.95 plus $1 postage and handling.

SOFTWARE

PSUEDO RANDOM NUMBER GENERATOR

by John D. Leasia, P.E.
2005 N. Wilson Ave.
Royal Oak, MI 48073

For a pseudorandom number generator that will generate all numbers from 00 to FF without skips or repeats, but without apparent pattern, try this after storing any two digit hex number seed in location 0000:

```
0200 D8       CLD     ; Clear decimal
0201 A5 00    LDA 00 ; Load seed = N
0203 0A 0A    ASL ASL; Multiply by 4 = 4N
0205 18       CLC     ;
0206 65 00    ADC 00 ; Add seed = 5N
0208 18       CLC     ;
0209 69 01    ADC# 01; Add 1 = 5N+1
020B 85 00    STA 00 ; Store seed
020D 60       RTS     ; Return
```

KIMATH SUPPORT

from John Eaton
1126 N 2nd
Vincennes IN 47591

Here's an applications program that uses KIMATH to find the TANGENT of any angle from 0 to 90 degrees. It converts the value in the RX register from Degrees to radians and uses the trigonometric identy listed in the KIMATH manual to find the TAN (RX). RX must be some value less than 90 degrees.

```
0300 20 7C FD  JSR CLRY
0303 A9 09     LDA# 09
0305 8D 48 02  STA SY+1
0308 A9 01     LDA# 01
030A 8D 58 02  STA EY      SET RY= 90
030D 20 16 FA  JSR DIVIDE Z=X/90
0310 20 0C FD  JSR MVZX
0313 20 5C FB  JSR TANX   Z=TAN(X/2), X is now in
                          radians
0316 20 0C FD  JSR MVZX
0319 20 10 FD  JSR MVZY
031C 20 08 F8  JSR ADD    Z=2TAN(x/2)
031F 20 14 FD  JSR MVZM   SAVE 2 TAN(X/2) in M reg-
                          ister
0322 20 0B F9  JSR MULTPY Z=(TAN(X/2))**2
0325 20 10 FD  JSR MVZY
0328 20 71 FD  JSR CLRX
032B A9 01     LDA# 01
032D 8D 36 02  STA SX+1 SET RX=1
0330 20 00 F8  JSR SUB   Z=1-(TAN(X/2))**2
0333 20 10 FD  JSR MVZY
0336 20 1C FD  JSR MVMX  X=2 TAN(X/2)
0339 20 16 FA  JSR DIVIDE Z=TAN(X)
```

The program uses the M register for temporary storage during processing. This should only be done during times that the other functions (LOG, TANX etc.) are not being used since they also use this register. By starting out in Degrees instead of radians we can get away from having to multiply the angle by a factor of 2/pi as shown in Appendix B of the KIMATH Manual.

This next routine is useful for those of us who hate to see s lot of trailing Zeros in an answer. Once KIMATH forms a result in the RZ register, this program will test it and set the value in PREC to be just large enough to cover the Non-zero digits. Placing this routine in your program before using the USTRESS, or PSTRESS routines will assure that you get all of the result and nothing more.

```
0300 A2 OF        LDX# OF
0302 BD 5A 02 LOP LDA RZ+1,X
0305 D0 03        BNE EXT
0307 CA           DEX
0308 D0 F8        BNE LOP
030A E8       EXT INX
030B 86 10        STX PREC
```


INTERRUPT ROUTINES AND BREAKPOINT

Markus Goenner
Switzerland

The three routines have all the same purpose, they decide if the occurred interrupt is generated form the soft- or hard-ware side. I do not mean the non-maskable interrupt. The break command (BRK) as well as the hardware int. forces an indirect jump via the vector at FFFE (17FE in the KIM-1 system) to the same interrupt routine. If the int. was caused by break command, the break flag is now set. We can use this fact to jump on a specific break routine (SWI=software int.) or an interrupt service routine (USINT=user int.). See minimum version.

The second routine uses the system monitor in case of a break command, but with the program counter adjusted to the breakpoint location (see lines 046...050). both of those routines are for people without a terminal.

The ultimate routine is for the Telet-pers and the Hexadisplayers as well. The vector for the non-maskable interrupt is $6C00 and $6C0C for the hardware int. If you work only with the hexadisplay, you may omit the lines higher than 092.

This routine is one of the best tools for software-debugging. You may set as many breakpoints (00) as you want. If the program reaches one, it will print all the registers and asks you for the byte which is replaced by the break command. The program starts from the point until the following breakpoint is encountered (if ever!)

You may hit the stop key on your hexa-keyboard in case of loosing control over the program. The program counter now points to the exact location where the stop occurred.

```
001          6502   KIM-1 IRQ-ROUTINE
002          ************************
003
004          (C) BY MARKUS P.GOENNER
005                BUEL
006                3205 HAUSS
007                SWITZERLAND
008
009          FEBRUARY,10 1978
010          *SIGYPHOS*PSEUDO-ASSEMBLER
011
012          MINIMUM VERSION
013          <><><><><><><><
014                        ;
015                        ;
016  0000                        *=$0200
017  0200  85 F3       IRQENT STA ACC
018  0202  68                   PLA
019  0203  48                   PHA
020  0204  29 10                AND #$ 10
021  0206  D0 05                BNE BREAK
022  0208  A5 F3                LDA ACC
023  020A  6C F5 17             JMP (USINT)
024  020D  6C F3 17    BREAK    JMP (SWI)
025                        ;
026                        ;
027                        ;
028                        ;
029                        ;         COMFORTABLE
030                        ;         <><><><><><
031                        ;
032                        ;
033  0000                        *=$0200
034  0200  85 F3       IRQENT STA ACC
035  0202  68                   PLA
036  0203  48                   PHA
037  0204  29 10                AND #$ 10
038  0206  D0 05                BNE BRKCMD
039  0208  A5 F3                LDA ACC
040  020A  6C FC 17             JMP (UEINT)
041  020D  68          BRKCMD   PLA
042  020E  85 F1                STA PREC
043  0210  D8                   CLD
044  0211  18                   CLC
045  0212  68                   PLA
046  0213  69 FF                ADC #$ FF
047  0215  85 EF                STA PCL
048  0217  85 FA                STA POINTL
```

```
049  0219  68              PLA
050  021A  69 FF           ADC #$ FF
051  021C  4C 0B 1C        JMP SAVE1+6
052                        ;
053                        ;
054                        ;
055                        ;
056                        ;       REAL DELUXE
057                        ;       <><><><><><>
058                        ;
059                        ;
060  0000                  *=$6C00
061  6C00  85 F3     NMIENT STA ACC
062  6C02  68               PLA
063  6C03  85 F1            STA PREG
064  6C05  68               PLA
065  6C06  85 EF            STA PCL
066  6C08  68               PLA
067  6C09  4C 26 6C         JMP STORE
068  6C0C  85 F3     IRQENT STA ACC
069  6C0E  68               PLA
070  6C0F  48               PHA
071  6C10  29 10            AND #$ 10
072  6C12  D0 05            BNE BRKCMD
073  6C14  A5 F3            LDA ACC
074  6C16  6C FC 17         JMP (USINT)
075  6C19  68        BRKCMD PLA
076  6C1A  85 F1            STA PREG
077  6C1C  D8               CLD
078  6C1D  18               CLC
079  6C1E  68               PLA
080  6C1F  69 FD            ADC #$ FD
081  6C21  85 EF            STA PCL
082  6C23  68               PLA
083  6C24  69 FF            ADC #$ FF
084  6C26  85 F0     STORE  STA PCH
085  6C28  84 F4            STY YREG
086  6C2A  86 F5            STX XREG
087  6C2C  BA               TSX
088  6C2D  86 F2            STX SPUSER
089  6C2F  A9 01            LDA #$ 01
090  6C31  2C 40 17         BIT SAD
091  6C34  F0 03            BEQ TTY
092  6C36  4C DC 1C         JMP PCCMD
093  6C39  20 2F 1E  TTY    JSR CRLF
094  6C3C  A5 F0            LDA PCH
095  6C3E  20 3B 1E         JSR PRTBYT
096  6C41  A5 EF            LDA PCL
097  6C43  20 3B 1E         JSR PRTBYT
098  6C46  A2 41            LDX #'A
099  6C48  20 B5 6C         JSR PRINT
100  6C4B  A5 F3            LDA ACC

101  6C4D  20 3B 1E         JSR PRTBYT
102  6C50  A2 58            LDX #'X
103  6C52  20 B5 6C         JSR PRINT
104  6C55  A5 F5            LDA XREG
105  6C57  20 3B 1E         JSR PRTBYT
106  6C5A  A2 59            LDX #'Y
107  6C5C  20 B5 6C         JSR PRINT
108  6C5F  A5 F4            LDA YREG
109  6C61  20 3B 1E         JSR PRTBYT
110  6C64  A2 53            LDX #'S
111  6C66  20 B5 6C         JSR PRINT
112  6C69  A5 F2            LDA SPUSER
113  6C6B  20 3B 1E         JSR PRTBYT
114  6C6E  A2 09            LDX #$ 09
115  6C70  BD CC 6C  PRILP  LDA TAB,X
116  6C73  20 A0 1E         JSR OUTCH
117  6C76  CA               DEX
118  6C77  10 F7            BPL PRILP
119  6C79  A5 F1            LDA PREG
120  6C7B  85 FC            STA TEMP
121  6C7D  A2 02            LDX #$ 02
122  6C7F  20 C1 6C  STATLP JSR STATUS
123  6C82  CA               DEX
124  6C83  D0 FA            BNE STATLP
125  6C85  06 FC            ASL TEMP
126  6C87  A9 2A            LDA #'*
127  6C89  20 A0 1E         JSR OUTCH
128  6C8C  A2 05            LDX #$ 05
129  6C8E  20 C1 6C  STLP   JSR STATUS
130  6C91  CA               DEX
131  6C92  D0 FA            BNE STLP
132  6C94  A2 50            LDX #'P
133  6C96  20 B5 6C         JSR PRINT

134  6C99  A5 F1            LDA PREG
135  6C9B  20 3B 1E         JSR PRTBYT
136  6C9E  29 10            AND #$ 10
137  6CA0  D0 06            BNE BREAK
138  6CA2  20 2F 1E         JSR CRLF
139  6CA5  4C 64 1C         JMP CLEAR
140  6CA8  A2 3F     BREAK  LDX #'?
141  6CAA  20 B5 6C         JSR PRINT
142  6CAD  20 9D 1F         JSR GETBYT
143  6CB0  91 EF            STA (PCL),Y
144  6CB2  4C C8 1D         JMP GOEXEC
145  6CB5  20 9E 1E  PRINT  JSR OUTSP
146  6CB8  8A               TXA
147  6CB9  20 A0 1E         JSR OUTCH
148  6CBC  A9 3D            LDA #'=
149  6CBE  4C A0 1E         JMP OUTCH
150  6CC1  06 FC     STATUS ASL TEMP

151  6CC3  A9 30            LDA #'0
152  6CC5  90 02            BCC OUTPUT
153  6CC7  A9 31            LDA #'1
154  6CC9  4C A0 1E  OUTPUT JMP OUTCH
155  6CCC  3D        TAB    '=
156  6CCD  43               'C
157  6CCE  5A               'Z
158  6CCF  49               'I
159  6CD0  44               'D
160  6CD1  42               'B
161  6CD2  2A               '*
162  6CD3  56               'V
163  6CD4  4E               'N
164  6CD5  20               ' 
```

SAMPLE RUN UNDERLINED DATA MEANS USER INPUT

```
KIM
0000 4C
0001 4F
0002 1C
0000 4C 00.
0001 4F
0000 00 G
0000 A=01 X=00 Y=FF S=F7 NV*BDIZC=11*10000 P=F0 ?=4C
KIM
0000 4C
1E63 A=01 X=00 Y=FF S=F5 NV*BDIZC=11*00110 P=E6
```

SQUARE-WAVER II

by Doug Jordan

A short SQUARE-WAVE program by Slagle was published in KUN:I(5)10, but this one is only half as long. Frequency is controlled by 00 0B and 00 0D. Requires audio output at PA0.

```
00   A9 01           LDA # 01      Set up PA0....
02   8D 01 17        STA   PADD    ... for output
05   49 01     ONE   EOR # 01      Flip output
07   8D 00 17        STA   PAD
0A   A9 xx           LDA # xx      Fetch frequency value
0C   8E 05 17        STA   CLK 8 T Set timer
0F   AE 07 17  TWO   LDA   CLK RD I Read timer
12   10 FB           BPL   TWO     Loop 'til time up
14   30 EF           BMI   ONE     Else loop back to toggle
16   00        END   BRK
```

KANSAS CITY COMPATABILITY

by Doug Jordan

Over a year has gone by and no one seems to have noticed the (anonymous ?) letter in the August 1977 Interface Age describing reading Kansas City Standard tapes by the unmodified KIM-1!! (vol. 2, no.9, p.9)

REVIEWS ETC.

BOOK REVIEW from the Editor

'6502 Applications Book' written by Rodney Zaks

The first thing I do with a new book is flip through the pages to get an initial reaction to its content.

My initial reaction to the '6502 Applications Book' was quite favorable in light of what I saw. A treatment of the family I/O chips, a touch tone dialer routine that used software to generate the frequencies, a morse code keyboard, a number of "quickie" interfaces and plenty of tidbits to while away the hours with.

Since I have been interested in telephone interfaces, I quickly "zeroed-in" on the touch-tone dialer program in the hopes of getting it running on my system.

One thing soon became apparent. The text mentioned that two timers would be necessary to generate the tones which would then be somehow "mixed" before going to a speaker, but there was no mention of what kind of speaker interface was necessary.

The program listing mentioned that the speaker would be hooked up in "configuration 2" but a search through the entire book failed to bring to the light of day the mysterious "configuration 2". A rank beginner would become totally frustrated.

The sobriety of the situation was lightened somewhat when I rediscovered the op-amp circuit that was presented at the end of the section as a hardware "improvement" for cleaner frequencies. "Improvement over what?" I wondered. The problem with this hardware "improvement" is that none of the parts values were indicated, not even the number of the op-amp.

As I was later to find, this "lack of attention to detail and lack of technical correctness" on the part of Zaks turned out to be the rule rather then the exception.

For instance, in another section of the book that supposedly deals with the circuitry necessary to drive relays from your computer, a circuit is shown to drive a +5 volt relay with a 7404 inverter. The very next drawing shows the schematic of a +12 volt relay with no mention of the fact that the 7404 inverter shown in the previous drawing will in no way drive a +12 volt relay (in this instance, it's assumed you're using a SYM with its built in high voltage driver capability). No mention is made of a circuit which would enable KIM or AIM to drive a +12 volt relay was ever made. (A simple circuit using an open collector driver such as the 7406 would have done the job.)

Again, very confusing for the beginner. I can't recommend this book.

Eric

POSTSCRIPT TO REVIEW: PROGRAMMING THE 6502 (Rodnay Zaks, SYBEX)

I have recently received the current Erratum sheet for this book. It contains well over 70 corrections.

Some of the corrections are relatively minor, being corrections in spelling, grammar, or wording. Others completely change the sense of the text, changing "left" to "right", or "it is possible..." to "it is not possible...". There are a few minor typographical errors in the corrections themselves, but they should not give the reader any problems.

The Erratum sheet corrects most of the errors of fact I have noticed in examining the book. I am still not happy about the book's approach to the subject; even with the "mechanics" corrected, it does a poor job of showing the reader how to apply the various coding techniques to solve a given programming problem.

If you have the book, you should write Sybex and ask for the erratum sheet, revision 1.1.

The erratum also notes that "a revised and expanded edition will be available shortly". I sincerely hope that the new edition is a major improvement over the old one.

--Jim Butterfield

PRODUCT REVIEW by Chuck Carpenter
2228 Montclair Pl
Carrollton TX 75006

MIMIC MICROCOMPUTER BOARD

MIMIC is a compact minimum microcomputer system. The unit has some expansion capability (an additional 128 bytes on board, a 30 pin buss external) and uses the 6500 family of microprocessors. A 45 page manual with information about the MIMIC system, data on the 6500 series microprocessor and operating instructions is included. The manual assumes you have prior knowledge of 6500 instructions for programming (or will get it from other sources).

My unit was purchased as a kit. The parts include a well-made circuit board, 10 I.C.'s including a 6504 microprocessor and a 6810 128 byte RAM, the usual variety of resistors and capacitors, 14 push button switches and 9 LED's. The switches and LED's make up the "front panel". Power can be supplied from a 6 volt lantern battery. I used about an hour to assemble and test MIMIC.

Assembly instructions are minimal and require a knowledge of electronic components and termin0 ology. No problem for anyone with a ham license and a more than casual interest. MIMIC can also be purchased assembled for about $65.

Programming is strictly in binary through the fromt panel switches. A unique latching arrangement lets you load addresses and then the data to be stored. The contents of any address can be examined at any time. I made up a form to allow hand assembly of programs and conversion to binary prior to entry. This simplified the address entry and data loading procedure. Six other switches are used for operation and control.

Writing programs relative to the stack, program counter (start vector) and interrupt vector are the responsibility of the user. In most other systems, these things are taken care of by the system monitor. It's not a problem and will certainly sharpen your programming skills. A memory map of the RAM used in your program can help keep you out of trouble. Programs are provided in the manual to help you get started. Remember: with 8 bits you can directly address only 256 bytes of memory.

I found MIMIC to be a well implemented circuit design and hardware assembly. Several mistakes, typo's and mis-information in the manual will confuse the neophyte programmer. However, MIMIC can provide a low cost source for learning the "innards" of a microcomputer. In fact, the only way you can talk directly to MIMIC is in 6500 binary: the processors native language. And MIMIC has utility value too. When you're through learning about the unit, you can turn it into a controller for your thermostat or other gadget project.

MIMIC can be obtained from Real Time Intelligence Corp., PO Box 9562, Rochester, N.Y. 14604. The kit price is $50.00. They appear to be a conscientious organization to deal with. Response has been excellent. I've enjoyed getting down to fundamentals with my MIMIC. I'm sure you will too.

BUGS

In Issue #15 I published a letter from Leo Jacobson in which it was stated that the National Bureau of Standards had purchased 29 Pets and was having trouble getting Commodore to service them.

I learned later that Mr. Jacobson had apparently been misinformed of the situation at the NBS and at his local Computerland store. Please disregard his comments and accept my apology for not checking the facts a little more closely.

Eric

In Issue #16, two boo-boos were found by sharp readers. I really goofed the Focal cassette interface on page 15. In line 0150 of the listing, HYPER should be addressed to $C400 (as in paragraph 4, page 15) not $0200. Also in that same listing, line 290 should read JMPCOM ***+3 (not JMPCOM ***+2). That and the missing RTS instruction after line 470 through the whole thing off. Here's a hex dump of the corrected program.

```
35EB  20 A3 29 20 AC
35F0  1F 20 A3 29 20 AC 1F A5 F8 8D F9 17 A9 4C 85 00

3600  A9 00 85 F1 60 20 EB 35 A5 31 8D F5 17 A5 32 8D
3610  F6 17 A5 3E 8D F7 17 A5 3F 8D F8 17 A9 00 85 01
3620  A9 20 85 02 4C 00 02 AD ED 17 85 3E AD EE 17 85
3630  3F 4C 00 20 20 EB 35 A9 27 85 01 A9 36 85 02 4C
3640  73 18 A9 4C 85 00 20 85 2F 85 01 A5 82 85 02 A9
3650  00 85 03 20 7B 36 84 04 20 7B 36 B4 05 20 7B 36
3660  A5 03 F0 0E C9 01 F0 08 C9 02 F0 09 C9 03 F0 05
3670  A5 04 4C 00 00 A5 05 AA 4C 70 36 A5 2F C9 2C F0
3680  04 C9 29 F0 06 20 7B 2F A8 E6 03 60 36 36 04 00
*
```

Also, on page 19 (issue #16) location $0B75 should be $80 (not $08) and location $0B9C should be $00 (not $AA).

Whew!!!!!!!!!!!!!

OOPS!!! I forgot to publish Bob Leedoms address in #16 (he wrote BASEBALL) so here it is-14069 Stevens Valley Ct, Glenwood MD 21738. I'm sure Bob would be glad to hear any comments you may have on his neat program.

CLUB NEWS

The San Fernando Valley KIM-1 Users Club has undergone a re-organization during the first part of the year. Jim Zuber, founder of the club, is no longer able to act as president due to an increased work load at his place of employment. Several changes have been made including a new name, new president, new meeting time and place, and new club organization. Here is the new information which you might want to publish in your excellent magazine:

NAME	The San Fernando Valley 6502 Users Club
TIME	2nd Tuesday of every month at 8:00 PM
PLACE	Computer Components of Burbank, Inc. 3808 West Verdugo Avenue, Burbank California 91505
CONTACT	Larry Goga, 3816 Albright Avenue, Los Angeles, California 90066 phone 213-398-6086
NEWSLETTER	published monthly at $2.00 per year
MEMBERSHIP	club is open to all owners of 6502 systems including AIM, SYM, KIM, APPLE, PET, etc.

Thank you once again for publishing your magazine. It is truly one of the finest publications in the area of personal computing.

CLUB ACTIVITIES IN DENMARK

A countrywide club covering 6502 microprocessor users in Denmark has been formed.

The club aims mostly at the users of basic systems such as KIM-1, SYM-1 and AIM-65, but other 6502 users are equally welcome to join in.

Although at present no membership fee is involved, several activities has been started:

1. Local meetings where project groups are established, publications are reviewed, and systems are described and demonstrated.
2. Publication of a newsletter, "MICROPOSTEN" which covers hardware design, software, product news and general information.
3. Establishment of a software library written by and for the members on a non profit basis.

The club is independent of commercial interests.

Any further information may be obtained from:

E. Skovgaard
Nordlundsvej 10
DK-2650 Hvidovre
Denmark

LET-TERS

Dear Eric:

Please add my name to what I hope is a growing list of those who have successfully copied J. C. Williams' 32K RAM design from User Notes #15. I do have some circuit changes that I strongly recommend, and some caveats.

First, damping resistors should be placed between the CAS, RAS, and WRITE drivers and the memory array to reduce undershoot on these signals. (This is common industry practice). I found that a value of 100 ohms was about optimum for my board. The value must be determined experimentally for each different layout, but most other builders will probably find that a value between 50 and 100 ohms will be correct.

Second, the provision the circuit makes to perform extra refresh cycles during system restart (i.e. powerup) may not be adequate to "wake up" some parts, most notably, older NEC (Nippon Electric Company) parts. These require 8 or so RAS-with-CAS (i.e. regular read or write) cycles after power-up before they function properly. Therefore, my system's restart routine, which is in PROM, does, among its other duties, 16 READs from each 16K bank, before attempting to use that memory.

Finally, passing a given memory test, even one that runs several hours, does not guarantee that the memory is working properly. Memory tests that exercise the memory continuously overlook some problems in 16K RAMs. Some parts, most notably older NEC parts again, have a problem unrelated to refresh that causes them to forget, temporarily, when they have not been accessed with a normal read or write cycle for a few milliseconds. Therefore, a good memory test for 16K RAMs is one that writes a pattern into the memory, waits several milliseconds, then reads back the pattern to verify it. Obviously, the memory test program may not be resident in the memory being tested because instruction fetches would keep the memory busy enough to mask the problem.

Sincerely,

Bob Haas
20887 SW Willupa Way
Tualatin OR 97062

6502 SOFTWARE

FORTH

* 6502 FORTH is a complete programming system which contains an interpreter/compiler as well as an assembler and editor.
* 6502 FORTH runs on a KIM-1 with a serial terminal. (terminal should be at least 64 chr. wide)
* All terminal I/O is funnelled through a jump table near the beginning of the software and can easily be changed to jump to user written I/O drivers.
* 6502 FORTH uses cassette for the system mass storage device
* Cassette read/write routines are built in (includes Hypertape).
* 92 op-words are built into the standard vocabulary.
* Excellent machine language interface.
* 6502 FORTH is user extensible.
* 6502 FORTH is a true implementation of FORTH according to the criteria set down by the FORTH Interest Group.
* Specialized vocabularies can be developed for specific applications.
* 6502 FORTH resides in 8K of RAM starting at $2000 and can operate with as little as 4K of additonal contiguous RAM.

6502 FORTH PRICE LIST

6502 FORTH SYSTEM ON KIM CASSETTE — $94.00
(includes user manual and annotated source listing for the $2000 version)
(also includes $4.00 for shipping and handling)

6502 FORTH USER MANUAL — $16.50
(full price is creditable towards FORTH software purchase)
(includes $1.50 for shipping and handling)

Our user manual assumes some previous knowledge of FORTH. If you have no idea what FORTH is all about-send a S.A.S.E. (business size) and ask for a "FORTH BIBLIOGRAPHY"

KIMATH

KIMATH ON CASSETTE OR EPROM FOR AIM, KIM, SYM, AND APPLE

STANDARD VERSIONS
KIMATH on KIM cassette (3x speed) — $12.00
(must specify $2000 or $F800 version) (includes errata sheet for manual)

CUSTOM VERSIONS
KIMATH is now available on EPROM or cassette assembled to any location and comes with a sorted symbol table for easy routine lookup.

On 3x KIM cassette — $20.00

On 2Kx8 EPROM (TI 2516 or Intel 2716) — $80.00
(APPLE version is only available on EPROM)

ORDERING INFORMATION FOR CUSTOM VERSIONS ONLY:

You must include the following information with your order for a custom version of KIMATH on KIM cassette or EPROM.

Hex starting address for main program (normally $F800)

Hex starting address for 23 bytes of zero-page storage (normally $0000)

Hex starting address for 154 bytes of RAM for the argument registers (normally $0200)

KIM SOFTWARE ON CASSETTE

FOCAL CASSETTE OPERATING SYSTEM
($4000-$4920) includes instructions, cassette and complete source listing. Price includes shipping & handling (works with either version of FOCAL) — $37.50

BASEBALL (from issue #16) — 6.00

BASEBALL source listing (16 pages) — 5.00

HEXPAWN (from issue #13) — 5.00

DISASSEMBLER (from issue #14) — 5.00

BANNER (from issue #14) — 5.00

These cassettes are original dumps, not copies, made with top quality 5-screw housing cassettes in the HYPERTAPE X3 tape speed. Thirty seconds of sync characters precede the program to enable you to tune up your recorder or PLL.

Payment must be in U.S. Funds. Overseas customers please include $1.00 extra per cassette for extra postage.

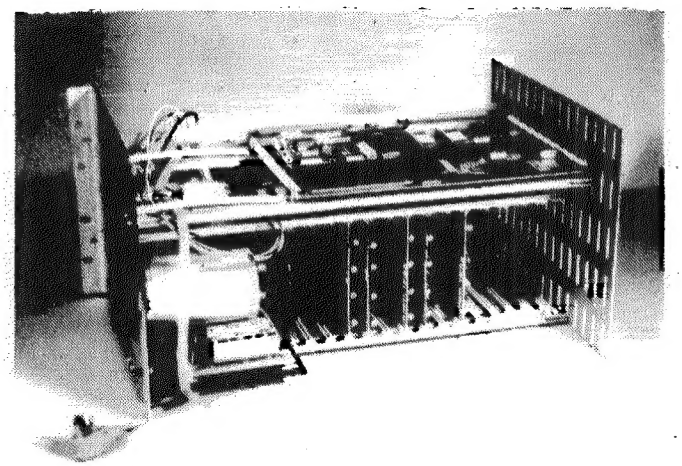
ORDER ALL 6502 SOFTWARE FROM:

ERIC C REHNKE
540 S RANCH VIEW CIR #61
ANAHEIM HILLS CA 92807

BOX 120
ALLAMUCHY, N.J. 07820
201-362-6574

HUDSON DIGITAL ELECTRONICS INC.

THE HDE CARD CAGE



Shown With KIM-1 (not included)

VERSIONS

KIM*	**AVAILABLE**
AIM*	1st Qtr. 80
SYM*	1st Qtr. 80

$525.00
Complete With Power Supply

Now you can expand your 65XX single board microcomputer into a powerful microprocessor based system with the 19" (RETMA standard) HDE DM816-CC15 Card Cage. The DM816-CC15 has virtually all of the features you need for even the most demanding situations. Complete with power supply, backplane, card guides and supports, the HDE DM816-CC15 accepts state of the art 4½" wide cards permitting your system to remain a compact configuration, while expanding with a variety of functions.

HDE has developed the DM816-CC15 for the demanding industrial marketplace. Consequently, you can design your KIM*, AIM* or SYM* based installation using RETMA standard cabinet or rack components. Sufficient clearance has been included for custom front panel switches, lights and controls as well as cable and fan installation at the rear. The microcomputer is mounted to permit convection cooling in all but the most densely packed situations.

The self-contained power supply is rated +8 VDC at 12 A and ±16 VDC at 3 A (both unreg.). The backplane, with the standard S44 bus, accepts up to 15 cards and has on board 5 VDC and 12 VDC regulators. In addition to power on reset, the backplane includes the logic connectors for remote reset stop and single step as well as cassette and 20 mA loop terminal I/O. Provisions for data and address bus termination are included. Two 16 pin DIP pads are available for unique requirements and the microcomputer application and expansion connectors are extended to the backplane further increasing the utility of the total package.

Other HDE products include:

- 5¼" and 8" single/dual disk systems
- 8K static RAM memory
- Prototyping cards
- Software (disk and cassette)
  - Text Editor (TED)
  - Text Output Processing System (TOPS)
  - Assembler (ASM)
  - Comprehensive Memory Test (CMT)
  - Dynamic Debugging Tool (DDT)

Watch for announcements:
EPROM Card, RS232 Card, PIA Card, DAC Card

* KIM Is a Commodore product
* AIM is a Rockwell International product
* SYM is a Synertec product

HDE PRODUCTS – BUILT TO BE USED WITH CONFIDENCE

AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

Johnson Computer
Box 523
Medina, Ohio 44256
(216) 725-4560

Plainsman Microsystems
Box 1712
Auburn, Alabama 36830
(800) 633-8724

ARESCO
P.O. Box 43
Audubon, Pa. 19407
(215) 631-9052

Long Island Computer General Store
103 Atlantic Ave.
Lynbrook, N.Y. 11563
(516) 887-1500

Lone Star Electronics
Box 488
Manchaca, Texas 78652
(512) 282-3570

Computer Lab of N.J.
538 Route 10
Ledgewood, N.J. 07852
(201) 584-0556